Chapter 3

3. Technology Adapted

3.1 Introduction

Detailed description of Existing System, weaknesses of the existing system and reasons for implementing the proposed software solution were discussed in the previous chapter.

Available technologies for Software Development Process, System Analysis and Designing, RDBMS etc that can be used for the proposed solution have been discussed in this chapter.

Methodologies are available for development of high quality software and this involves series of software development process aimed towards perfecting the software. Software development methodology is the whole process which is undergone in the development of software. Achieving end-user satisfaction is not a simple task indeed. Most software projects fail because of not using best practices in software development. Over 80% of projects are unsuccessful either because they are over budget, late, missing function, or a combination. Moreover, 30% of software projects are so poorly executed that they are cancelled before completion [1].

Not fulfilling of single requirement some times would lead to non compliance to the purpose for which the software was initially targeted. This will ultimately affect the end user un-satisfaction and hence leads to the failure in the whole project incurring major financial losses.

To overcome this situation, professionals have introduced, through their knowledge and experience, processors and methodologies which would streamline the software development process and improves communication, between the end user and the

development team bi-directionally, which will make sure that all the requirements of the end user are exactly met.

Although there are many software processes, some fundamental activities are common to all software processes as in below [2].

1. *Software specification* where customers and engineers define the software to be produced and the constraints on its operation.
2. *Software development* where the software is designed and programmed.
3. *Software validation* where software is checked to ensure that it is what the customer requires.
4. *Software evolution* where the software is modified to adapt it to changing customer and market requirement.

Since use of an inappropriate software process may reduce the quality or the usefulness of the software product to be developed and/or increase the development cost and the time, it is very much important to evaluate all the processes thoroughly and select the best suitable process.

3.2 Software Process Models

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective [2]

- The waterfall model

  Separate and distinct phases of specification and development
- Evolutionary development

  Specification and development are interleaved
- Reuse/component-based development

  The system is assembled from existing components

3.2.1 Waterfall Model

The waterfall model is a sequential software development model in which development is seen as flowing steadily downwards (like a waterfall) through several process phases.

3.2.1.1 Phases of the waterfall model

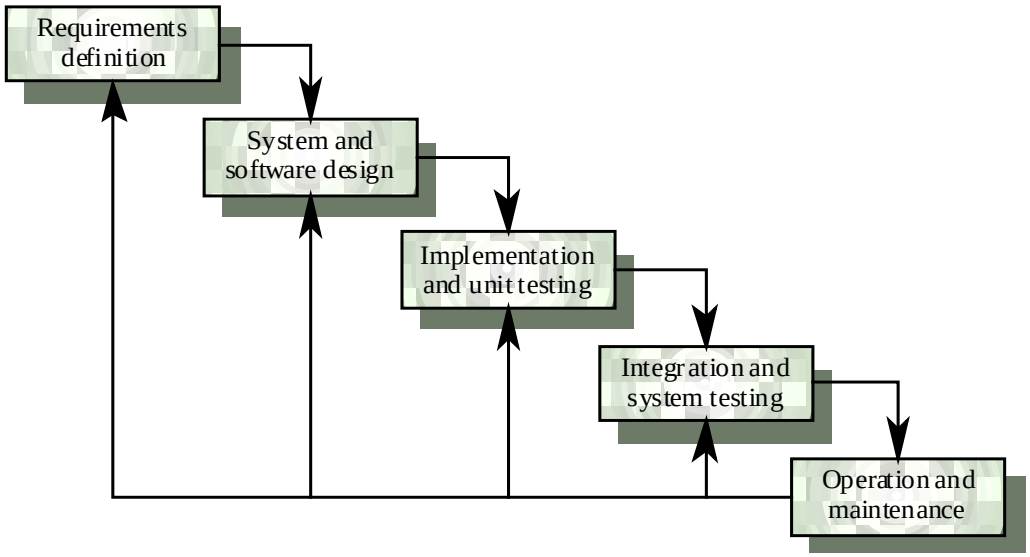

Figure 3.1: Phases of Waterfall Model

Requirements analysis and definition

All possible requirements of the system to be developed are captured in this phase. Requirements are set of functionalities and constraints that the end-user (who will be using the system) expects from the system. The requirements are gathered from the end-user by consultation. To understand what type of the programs to be built, the system analyst must study the information domain for the software as well as understand required function, behaviour, performance and interfacing.

These requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied. A Requirement Specification document which serves the purpose of guideline for the next phase of the model is created finally [3].

System and software design

Before starting for actual coding, it is highly important to understand what we are going to create and what it should look like? The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model [3].

Implementation and unit testing

On receiving system design documents, the work is divided in modules/units and actual coding is started. The system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality; this is referred to as Unit Testing. Unit testing mainly verifies if the modules/units meet their specifications [3].

Integration and system testing

As specified above, the system is first divided in units which are developed and tested for their functionalities. These units are integrated into a complete system during Integration phase and tested to check if all modules/units coordinate between each other and the system as a whole behaves as per the specifications. After successfully testing the software, it is delivered to the customer [3].

Operation and maintenance

This phase of "The Waterfall Model" is virtually never ending phase (very long). Generally, problems with the system developed (which are not found during the development life cycle) come up after its practical use starts, so the issues related to the system are solved after deployment of the system. Not all the problems come in picture directly but they arise time to time and needs to be solved; hence this process is referred as maintenance [3].

3.2.1.2 Advantages of Waterfall Model

- A schedule can be set with deadlines for each stage of development.
- This model emphasizes on the documentation at the end of each phase.
- Therefore, this model is only appropriate when the requirements are well-understood and no drastic changes into the requirements throughout the process.

3.2.1.3 Weaknesses of Waterfall method

- The entire functionality is developed and then tested all together at the end. Hence major design problems may not be detected till very late.
- Customers can not use anything until the entire system is complete.
- The model makes no allowances for prototyping.

- Difficulty of accommodating changes while the process is underway. This makes it difficult to respond to changing customer requirements.
- Difficult to establish all requirements explicitly, no room for uncertainty. This may lead to spending lot of resources at the beginning of the project; where as in findings are more obvious in the successive phases.
- Inflexible partitioning of the project into distinct stages, start of one phase should not take place before the signing off the documentation generated over the previous phase. This is contrary to the practical scenario of overlapping the said phases and involves a sequence of iterations at each of the phases.
- The iterations are performed only after a complete cycle and there by iterations are costly and involves a significant amount of work.

3.2.2 Evolutionary Development

"Evolutionary delivery is a lifecycle model that straddles the ground between evolutionary prototyping and staged delivery". An initial system is rapidly developed from abstract specifications. Early versions of the system are presented to the customer and then system is refined and enhanced based on customer feedback. The cycle continues until development time runs out (schedule constraint) or funding for development runs out (resource constraint) [4].

Evolutionary Development can be visualised as in below Figure 3.2.

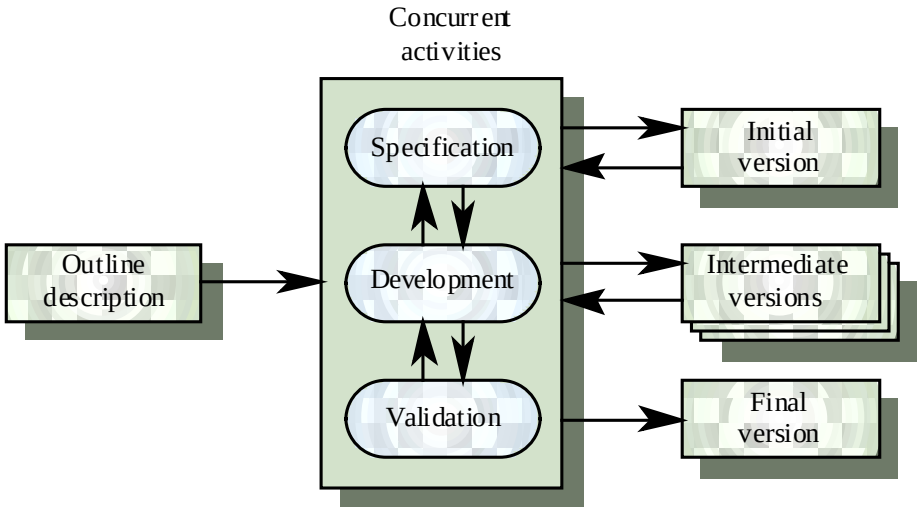

Figure 3.2: Evolutionary Development Process

Two main fundamental types are available under the evolutionary development as described below.

3.2.2.1 Exploratory prototyping

Objective is to work with customers and to evolve a final system from an initial outline specification. Starting has to be done with well-understood requirements and new features have to be added to the final system as proposed by the customer through a number of stages.

3.2.2.2 Throw-away Prototyping

A prototype which is usually a practical implementation of the system is produced to help discover requirements problems and then discarded. The system is then developed using some other development process. Objective is to understand the system requirements with poorly understood requirements and clarify what is really needed. This method is used to reduce requirements risk

3.2.2.3 Advantages of Evolutionary Development

- Rapid delivery and deployment.
- Users can experiment with a prototype to see how the system supports their work and hence risk can be minimised.
- Misunderstandings between software users and developers are exposed
- Missing services may be detected and confusing services may be identified early.
- A working system is available early in the process.
- Improved system usability is there, because from the beginning users comment about the usability by using the prototype.
- Because of changes in the external environment, require reshaping of the original business problem and these changes can be incorporated easily.
- The specifications can be developed incrementally as users develop a better understanding of their problem.

3.2.2.4 Weaknesses of Evolutionary Development

- Lack of process visibility, as all the phases are overlapping and are impractical to provide documentation for every version of the system.
- Systems are often poorly structured as addition of changes at each version of the system will not permit to have the best possible structure as in the case of pre-identifying all the requirements.
- Special skills may be required on RAD languages to develop prototypes.

3.2.3 Component-Oriented Development

Component-Oriented Development is concerned with the assembly of pre-existing software components into larger pieces of software. Significant number of reusable components should be existence. The system development process focuses integrating these components into a system rather than developing them from scratch [2]. The process is shown in Figure 3.3.

3.2.3.1 Process stages

While the initial requirements specification stage and the validation stage are comparable wit other process, the intermediate stages in a reuse-oriented process are different. These stages have been discussed below [2].

Component analysis

Given the requirement specification, a search is made for components to implement the specification. Usually there is no exact match and the components that may be used may only provide some of the functionality required.

Requirements modification

The requirements are analysed using the information about the components that have been discovered. They are then modified to reflect the variable components. Where modifications are impossible, the component analysis may be re-entered to search for alternative solutions.

System design with reuse

The framework of the system is designed or an existing framework is re-used. The designers take into account the components that are re-used and organise the

framework to cater to this. Some new software has to be designed if the components are not available.

Development and integration

The software that can't be externally produced is developed, COTS systems are integrated to create the new system. System integration, in this model, may be a part of the development process rather than a separate activity.

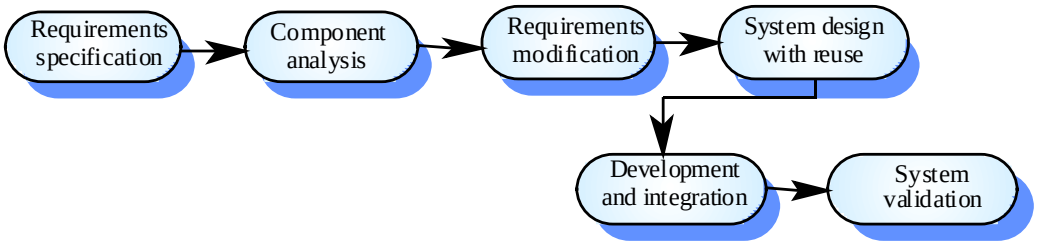

Figure 3.3: Component-Oriented Development Process

3.2.4 Process Iteration

Businesses now operate in a rapidly changing environment. Hence, requirement changes of business systems are inevitable. Iterative development slices the deliverable business value (system functionality) into iterations and this approach is a development approach that "cycles" through the development phases, from gathering requirements to delivering functionality in a working release in order to support for handling of changing requirements. In other words, *Iteration* refers to the cyclic nature of a process in which activities are repeated in a structured manner. There are two process models that have been explicitly designed support process iteration.

1. Incremental Development
2. Spiral development

3.2.4.1 Incremental development

Rather than delivering the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality. User requirements are prioritised and the highest priority requirements are included in early increments. Here *increment* refers to the quantifiable outcome of each iteration. Iteration Process is shown in Figure 3.4.

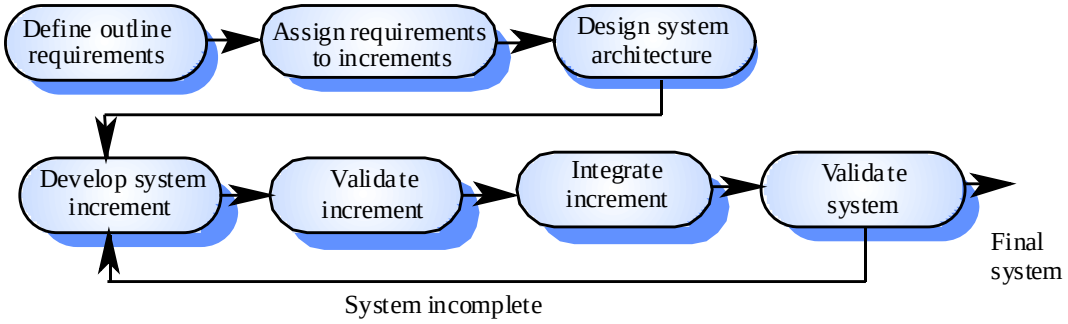

Figure 3.4: Iteration Process

3.2.4.2 Advantages of Incremental Developments

- Customer does not want to wait until the entire system is delivered. The first increment satisfies their most critical requirements and can gain the value from it.
- Early increments can be used as a prototype and customer can gain experience from it and can inform their requirements for later system increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing and hence, risk of failure of most important parts of the system is very less.

3.2.4.3 Weakness of the Incremental Development

- Increments have to be relatively small
- Each increment should deliver some functionality
- As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.

3.2.4.4 Spiral Development

The final evolution from the water fall is the spiral, taking advantage of the fact that development projects work best when they are both incremental and iterative, where the team is able to start small and benefit from enlightened trial and error along the way. This methodology reflects the relationship of tasks with rapid prototyping, increased parallelism, and concurrency in design and builds activities. The spiral method should still be planned methodically, with tasks and deliverables identified for each step in the spiral. This methodology can be visualised as in Figure 3.5.

The steps in the spiral model can be generalized as follows:

1. The new system requirements are defined in as much detail as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
2. A preliminary design is created for the new system.
3. A first prototype of the new system is constructed from the preliminary design and is a scaled down version of the final product.
4. A second prototype is evolved by a fourfold procedure:
    1. Evaluating the first prototype in terms of its strengths, weaknesses, and risks;
    2. Defining the requirements of the second prototype;
    3. Planning and designing the second prototype;
    4. Constructing and testing the second prototype.
5. At this point the customer may decide to scrap the whole project if the risk is too high.
6. Evaluate the current prototype in the same way as the previous prototype and create another one if needed
7. Iterate the proceeding steps until the customer is satisfied that the current prototype represents the final product.
8. Construct the final system
9. The final system is thoroughly evaluated and tested and routine maintenance is carried out for the life of the product.

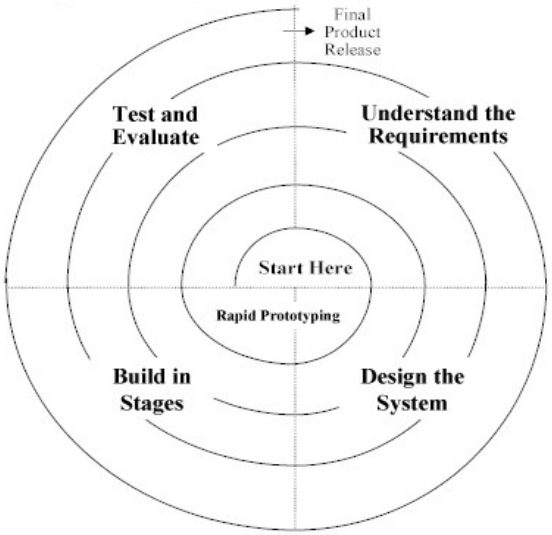

Figure 3.5: Spiral Development Model

3.2.4.5 Advantages of Spiral Development

- Software engineers can start working on the project earlier rather than wading through a lengthy early design process.
- Offers prototyping as a risk-reduction option at any stage of development.
- It allows reworks of earlier stages as more attractive alternatives are identified.
- It promotes reuse of existing software in early stages of development.
- Eliminates errors and unattractive alternatives early.
- It balances resource expenditure.

3.2.4.6 Weakness of Spiral Development

- Estimates of budget and time are harder to judge at the beginning of the project since the requirements evolve through the process.

3.2.5 Rapid Application Development (RAD)

"Rapid Application Development (RAD) is a development lifecycle designed to give much faster development and higher-quality results than those achieved with the traditional lifecycle. It is designed to take the maximum advantage of powerful development software that has evolved recently." [7].

Rapid application development is a software development methodology, which involves iterative development and the construction of prototypes. RAD thus enables quality products to be developed faster, saving valuable resources. The speed increases can be achieved using a variety of methods including, rapid prototyping, virtualization of system related routines, the use of CASE tools and other techniques.

Prototyping helps the analyst and users to verify the requirements and to formally refine the data and process models.

3.2.5.1. Advantages of Rapid Application Development

- ***Increased speed of development***

  As the name suggests, Rapid Application Development's primary advantage lies in an application's increased development speed and decreased time to delivery. The goal of delivering applications quickly is addressed through the use of Computer Aided Software Engineering or CASE tools, which focus on converting requirements to code as quickly as possible, as well as Time Boxing, in which

features are pushed out to future releases in order to complete a feature light version quickly. Development through rapid prototyping also contributes for fast delivery of the solution.

- ***Increased quality***

Increased quality is a primary focus of the Rapid Application Development methodology, but the term has a different meaning than is traditionally associated with Custom Application Development. Prior to RAD, and perhaps more intuitively, quality in development was both the degree to which an application conforms to specifications and a lack of defects once the application is delivered. According to RAD, quality is defined as both the degree to which a delivered application meets the needs of users as well as the degree to which a delivered system has low maintenance costs.

Rapid Application Development attempts to deliver on quality through the heavy involving of users in the analysis and particularly the design stages.

- Better end-user utility
- Reduced complexity of the application due to the usable GUI design and accustomed look and feel
- Clear communication with the customer ensuring what will be delivered and when.
- Lower Cost

3.2.5.2 Weakness of Rapid Application Development

- Reduced Scalability,

  Because RAD focuses on development of a prototype that is iteratively developed into a full system, the delivered solution may lack the scalability of a solution that was designed as a full application from the start.

- Reduced features

  Due to time boxing, where features are pushed off to later versions in favour of delivering an application in a short time frame, RAD may produce applications that are less full featured than traditionally developed applications.

- As it takes very little time & tasks are automated, so, confidence in product is usually low for risky and mission-critical applications.

The applicability of one or more process models will be discussed in Chapter 4, in detail.

3.3 Software Analysis and Design Methodologies

System analysis is an important activity that takes place when new information systems are being built. This is a systematic approach to

- identifying problems, opportunities, and objectives
- analyzing the information flows in organizations
- designing computerized information systems to solve a problem

Use of correct and clear System Analysis and Design methodology helps to close the communication gaps between business people and IS developers, as well as between developers and developers. This not only helps us ensure the system is built right but also that the right system is built for the business. Stages in building a software system has been illustrated in Figure 3.6. SSADM and Object Oriented System analysis and design methodologies have been discussed in the below part of this chapter.

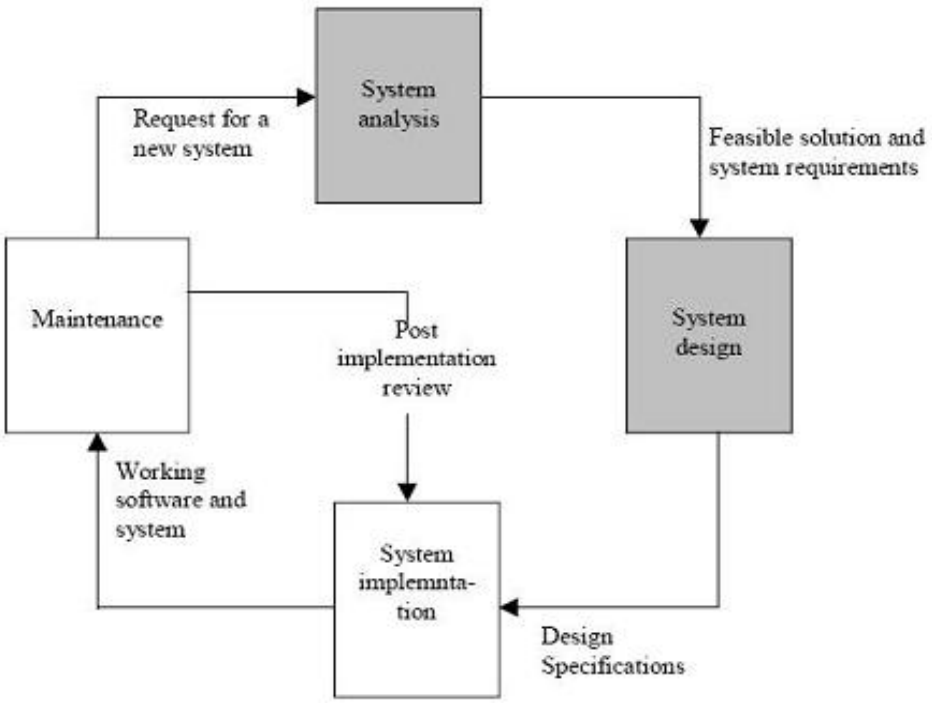

Figure 3.6 Stages in building a Software System

3.3.1 Structured Systems Analysis and Design Methodology (SSADM)

Structured **S**ystems **A**nalysis and **D**esign **M**ethodology, a set of standards developed in the early 1980s for systems analysis and application design widely used for computing projects. SSADM uses a combination of text and diagrams throughout the whole life cycle of a system design, from the initial design idea to the actual physical design of the application.

SSADM can be thought to represent a pinnacle of the rigorous document-led approach to system design. Further, SSADM is one particular implementation and builds on the work of different schools of development methods.
The 3 most important techniques/tools that are used in SSADM are:

Logical Data Modeling

This is the process of identifying, modeling and documenting the data requirements of the system being designed. The data are separated into entities and relationships.

Data Flow Modeling.

This is the process of identifying, modeling and documenting how data moves around an information system. Data Flow Modelling examines processes, data stores, external entities, and data flows.

Entity Behaviour Modeling

This is the process of identifying, modeling and documenting the events that affect each entity and the sequence in which these events occur.

SSADM application development projects are divided into five modules that are further broken down into a hierarchy of stages, steps and tasks:

1. **Feasibility Study** -- the business area is analyzed to determine whether a system can cost effectively support the business requirements.
2. **Requirements Analysis** -- the requirements of the system to be developed are identified and the current business environment is modeled in terms of the processes carried out and the data structures involved.

3. **Requirements Specification** -- detailed functional and non-functional requirements are identified and new techniques are introduced to define the required processing and data structures.
4. **Logical System Specification** -- technical systems options are produced and the logical design of update and enquiry processing and system dialogues.
5. **Physical Design** -- a physical database design and a set of program specifications are created using the logical system specification and technical system specification.

3.3.2 Object Oriented Systems Analysis and Design Methodology

There have been basically 3 approaches in information system development area: process-oriented, data-oriented and object-oriented approaches. As information technology (both hardware and software) has been advancing, people have moved from the earliest process-oriented approach to data-oriented approach and now begun to adopt the latest object-oriented analysis methodology.

Unlike its two predecessors that focus either on process or data, the object-oriented approach combines data and processes (called methods) into single entities called objects. Objects usually correspond to the real things an information system deals with, such as customers, suppliers, contracts, and rental agreements. Object-oriented model is able to thoroughly represent complex relationships and to represent data and data processing with a consistent notation, which allows an easier blending of analysis and design in an evolutionary process. The goal of object-oriented approach is to make system elements more reusable, thus improving system quality and the productivity of systems analysis and design [5].

Use-case modeling

Use-case modeling is developed in the analysis phase of the object-oriented system development life cycle. Use-case modeling is done in the early stages of system development to help developers gain a clear understanding of the functional requirement of the system, without worrying about how those requirements will be implemented.

A use-case is a representation of a discrete set of work performed by a use (or another system) using the operational system. A use-case model consists of actors and use cases. An actor is an external entity that interacts with the system and a use case represents a sequence of related actions initiated by an actor to accomplish a specific goal.

By asking the following questions, use cases can be identified:

- What are the main tasks performed by each actor?
- Will the actor read or update any information in the system?
- Will the actor have to inform the system about changes outside the system?
- Does the actor have to be informed about unexpected changes?

Use-cases are shown as ellipses with their names inside and are performed by the actors outside the system. A use-case is always initiated by an actor. A use-case may interact with other use-cases. Some of these relationships include extend and use and are reflected by single hollow arrow lines.

While a use-case diagram shows all the use cases in the system, it does not describe how those use cases are carried out by the actors. The contents of a use case are normally described in plain text. While describing a use case, it is required to focus on its external behaviour, that is, how is interacts with the actors, rather than how the use case is performed inside the system

One of the benefits of use-case is its simplicity. The strength of the technique is in its non-technical simplicity, which allows users to participate in a way that is seldom possible using the abstractions of Class Modeling alone. It also helps the analyst get to grip with specific user needs before analyzing the internal mechanics of a system. Use cases also fit particularly well within an evolutionary and incremental process in that they provide a basis for early prototyping and readily identifiable units for incremental delivery. They also provide a means of traceability for functional requirements upstream in the process and for constructing test plans downstream in the process. In summary, use case model provides a complete list of the steps in system usage modeling as follows:

· Identify the actors.

· Identify the use cases.

· Create a Use Case Diagram.

· Describe the use cases.

· Complete the use case descriptions.

Class Modeling

An object is the most fundamental element in OO approach, which has a well-defined role in the application domain, and has state, behaviour, and identity. A class is a set of objects that share the same attributes, operations, methods, relationships, and semantics. A class may use a set of interfaces to specify collections of operations it provides to its environment.

Object modeling, or class modeling is the key activity in object-oriented development. If the use cases contain errors, then all is not lost. If the class model contains errors then all may well be lost. The quality of the resulting system in object-oriented development is essentially a reflection of the quality of the class model. This is because the class model sets the underlying foundation upon which objects will be put to work. A quality class model should provide a flexible foundation upon which systems can be assembled in component-like fashion. A poor class model results in a unstable foundation upon which systems will grind to a halt and buckle under the threat of change.

Each of the diagrams used in UML lets you see a business process from a different angle. Business users, for example, can view use case diagrams to see the business scenario overview and understand who's doing what, while developers can use class and object diagrams to get accurate descriptions of how to build those components into their code. The class and object diagrams are so detailed, describing elements such as interfaces and attributes, that translating UML notation into actual programming code is a virtual no-brainer.

In UML, a class is represented by a rectangle with three compartments separated by horizontal lines, which hold, from top to bottom, class name, the list of attributes, and the list of operations.

A class diagram allows to document how the class relates to other classes. The class diagram doesn't fix the actual implementation. The actual code might not be a direct translation of the diagram. However the functionality of the code will remain the same.

3.3.3 Unified Modeling Language (UML)

The Unified Modeling Language (UML) is an object-oriented language for specifying, visualizing, constructing, and documenting the artefact of software systems, as well as for business modeling (UML Document Set, 2001). The UML was developed by Rational Software and its partners.

Because UML uses simple, intuitive notation, nonprogrammers can also understand UML models. In fact, many of the language's supporters claim that UML's simplicity is its chief benefit. Since developers, customers, and implementers can all understand a UML diagram, they are more likely to agree on the intended functionality, thereby improving their chances of creating an application that truly solves a business problem.

The UML, a visual modeling language, is not intended to be a visual programming language. The UML notation is useful for graphically illustrating of object-oriented analysis and design models. It not only allows us to specify the requirements of a system and capture the design decisions, but it also promotes communication among key persons involved in the development effort.

There are total nine types of UML diagrams: class, object, use case, sequence, collaboration, state-chart, activity, component, and deployment. While a use-case diagram shows the interaction between external actors and actions performed within a system, a class diagram describes the static structure of objects in the problem domain. The other seven diagrams all provide differing perspectives on systems and are also useful in system analysis and design.

3.4 Database Technology

3.4.1 Relational Database Management System (RDBMS)

A **Relational database management system (RDBMS)** is a database management system (DBMS) that is based on the relational model. RDBMS data is structured in database tables, fields and records. Each RDBMS table consists of database table rows. Each database table row consists of one or more database table fields. Further, RDBMS provides relational operators to manipulate the data stored into the database tables. Most popular commercial and open source databases currently in use are based on the relational model.

RDBMS is a program that can be used to create, update, and administer a relational database. Most commercial RDBMSs use the Structured Query Language (SQL) as database query language. Further, RDBMS usually include an API so that developers can write programs that use them.

The most popular RDBMS are Microsoft Access, MS SQL Server, DB2, Oracle and MySQL.

3.4.2 Normalizing Databases

Normalization is the process of efficiently organizing data in a database. There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table). Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.

Hence, applying the principles of normalization to the database design will help to improve the performance of the DBMS drastically.

The Normal Forms

There are four normal forms that can be applied for a RDBMS. The database community has developed a series of guidelines for ensuring that databases are normalized. These are referred to as normal forms and are numbered from one (the lowest form of normalization, referred to as first normal form or 1NF) through five

(fifth normal form or 5NF). Mostly, only 1NF, 2NF, and 3NF along with the occasional 4NF can be seen in practical applications. Fifth normal form is very rarely used in the applications.

First Normal Form (1NF)

First normal form (1NF) sets the very basic rules for an organized database:

- Eliminate duplicative columns from the same table.
- Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

Second Normal Form (2NF)

Second normal form (2NF) further addresses the concept of removing duplicative data:

- -Meet all the requirements of the first normal form.
- Remove subsets of data that apply to multiple rows of a table and place them in separate tables.
- Create relationships between these new tables and their predecessors through the use of foreign keys.

Third Normal Form (3NF)

Third normal form (3NF) goes one large step further:

- Meet all the requirements of the second normal form.
- Remove columns that are not dependent upon the primary key.

Fourth Normal Form (4NF)

Finally, fourth normal form (4NF) has one additional requirement:

- Meet all the requirements of the third normal form.
- A relation is in 4NF if it has no multi-valued dependencies.

These normalization guidelines are cumulative. For a database to be in 2NF, it must first fulfil all the criteria of a 1NF database.

3.4.3 Securing Databases

Information security is a critical element of business operations and in order to protect the information assets against internal and external threats, it is very much important to implement a very strong security system for the database. Further,

database security mechanisms protect the confidentiality, integrity and availability of organization's most sensitive data.

Securing database schemas by limiting their privileges, providing good password support, restricting access using multiple defences, and securing the network channels to and from the database are some of the techniques that can be implemented with many of the best practices in order to have a better information security system.

3.5 Windows Forms and Web Forms

In developing applications with a user interface, either Windows Forms or Web Forms can be used. Both have full design-time support within the development environment, and can provide a rich user interface and advanced application functionality to solve business problems. Because of this feature parity, sometimes it is difficult to decide which technology is most appropriate for a given application.

Certain application considerations might make the choice obvious. For example, in creating an e-commerce Web site that will be accessible to the public over the Internet, Web Forms pages will give better performance and results. If the application is a processing-intensive, highly responsive application that needs the full functionality of the client computer, such as an office productivity application then the Windows Forms is ideal. However, in other cases the choice might not be so clear [6].

3.5.1 When to Use Windows Forms

Windows Forms are better when it is needed the client application to be responsible for much of the processing burden in an application. These client applications include Win32 desktop applications that were traditionally developed in previous versions of Visual Basic and Visual C++. Examples include drawing or graphics applications, data-entry systems, point-of-sale systems, and games.

These applications all rely on desktop computer power for processing and high-performance content display. Some Windows Forms applications might be entirely self-contained and perform all application processing on the user's computer. Other applications might be part of a larger system and use the desktop computer primarily

for processing user input. For example, a point-of-sale system often requires a responsive, sophisticated user interface that is created on the desktop computer, but is linked to other components that perform back-end processing.

In Windows Forms Windows applications which are built around a Windows framework, the application has access to system resources on the client computer, including local files, the Windows registry, the printer, and so on. This level of access can be restricted to eliminate any security risks or potential problems that arise from unwanted access. Additionally, Windows Forms can use the .NET Framework GDI+ graphics classes to create a graphically rich interface, which is often a requirement for data-mining or game applications.

3.5.2 When to Use Web Forms

Naturally if the application is intended to be available publicly through the World Wide Web, such as e-commerce applications, Web forms are ideal. But Web Forms can be used to create more than just Web sites; many other applications lend themselves to a "thin front end" as well, such as Internet-based employee handbook or benefits applications. An important benefit of any Web Forms application is that it has no distribution costs. Users have already installed the only piece of the application that they need the browser.

Web Forms applications are platform-independent that is, they are "reach" applications. Users can interact with the application regardless of what type of browser they have and even what type of computer they are using. At the same time, Web Forms applications can be optimized to take advantage of features built into the most recent browsers, such as Microsoft Internet Explorer 7.0, to enhance performance and responsiveness.

3.6 Summary

The chapter provided a literature survey of some of the widely used process models and System Analysis and Design Methodologies. Apart from that, it has explained about the Data Base Management Systems, Data security and programming techniques. The following chapter discusses about selecting of the right methodology for the proposed system with justifications.