A Software requirements specification
On

Image Recognition System

Submitted in Partial Fulfillment for the Award of Degree of Bachelor of Technology in Computer Science and EngineeringfromRajasthan Technical University, Kota

MENTOR:

Mrs Kajal Mathur
(Dept. of Computer Science & Engineering)

SUBMITTED BY:

Mohit Jain (19ESKCS144)

Navdeep Dhakar (19ESKCS157)

COORDINATOR:

Ms. Abha Jain
(Dept. of Computer Science & Engineering)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SWAMI KESHWANAND INSTITUTE OF TECHNOLOGY, MANAGEMENT & GRAMOTHAN
Ramnagaria (Jagatpura), Jaipur – 302017

SESSION 2022-23

TABLE OF CONTENTS

Introduction

Image recognition software is based on the ability to recognize a face and then measure the various features of the face. This project is aimed to identify the face of the person using various features like eyes, hair, lips, nose, etc. The details such as distance between the eyes or shape of the chin, are then converted into a mathematical representation and compared to data on other faces collected in a face recognition database.

- **Face Detection:** Look at the picture and find a face in it.
- **Data Gathering:** Extract unique characteristics of Aditya's face that it can use to differentiate him from another person, like eyes, mouth, nose, etc.
- **Data Comparison:** Despite variations in light or expression, it will compare those unique features to all the features of all the people you know.
- **Face Recognition:** It will determine "Hey, that's my boy Aditya!

Scope and Objective

PROJECT SCOPE:

The scope of the project is confined to store the image and store in the database. When a person has to be identified the images stored in the database are compared with the existing details. Over the last ten years or so, face recognition has become a popular area of research in computer vision and one of the most successful applications of image analysis and understanding. Because of the nature of the problem, not only computer science researchers are interested in it, but neuroscientists and psychologists also. It is the general opinion that advances in computer vision research will provide useful insights to neuroscientists and psychologists into how human brain works, and vice versa.

Image Recognition systems use computer algorithms to pick out specific, distinctive details about a person's face. These details, such as distance between the eyes or shape of the chin, are then converted into a mathematical representation and compared to data on other faces collected in a face recognition database. The data about a particular face is often called a face template and is distinct from a photograph because it's designed to only include certain details that can be used to distinguish one face from another.

PROJECT OBJECTIVE:

This project is intended to identify a person using the images previously taken. The identification will be done according the previous images of different persons. Every face has numerous, distinguishable **landmarks**, the different peaks and valleys that make up facial features. It defines these landmarks as **nodal points**. Each human face has approximately 80 nodal points. Some of these measured by the software are:

- Distance between the eyes
- Width of the nose
- Depth of the eye sockets
- The shape of the cheekbones
- The length of the jaw line

These nodal points are measured creating a numerical code, called a **faceprint**, representing the face in the database.

System Analysis

The first step in developing anything is to state the requirements. This applies just as much to leading edge research as to simple programs and to personal programs, as well as to large team efforts. Being vague about your objective only postpones decisions to a later stage where changes are much more costly.

The problem statement should state what is to be done and not how it is to be done. It should be a statement of needs, not a proposal for a solution. A user manual for the desired system is a good problem statement. The requestor should indicate which features are mandatory and which are optional, to avoid overly constraining design decisions. The requestor should avoid describing system internals, as this restricts implementation flexibility. Performance specifications and protocols for interaction with external systems are legitimate requirements. Software engineering standards, such as modular construction, design for testability, and provision for future extensions, are also proper.

Many problems statements, from individuals, companies, and government agencies, mixture requirements with design decisions. There may sometimes be a compelling reason to require a particular computer or language; there is rarely

justification to specify the use of a particular algorithm. The analyst must separate the true requirements from design and implementation decisions disguised as requirements. The analyst should challenge such pseudo requirements, as they restrict flexibility. There may be organizational reasons for the user requirements, but at least the analyst should recognize that these externally imposed design decisions are not essential features of the problem domain.

A problem statement may have more or less detail. A requirement for a conventional product, such as a payroll program or a billing system, may have considerable detail. A requirement for a research effort in a new area may lack many details, but presumably the research has some objective, which should be clearly stated.

Most problem statements are ambiguous, incomplete, or even inconsistent. Some requirements are just plain wrong. Some requirements, although precisely stated, have unpleasant consequences on the system behaviour or impose unreasonable implementation costs. Some requirements seem reasonable at first but do not work out as well as the request or thought. The problem statement is just a starting point for understanding the problem, not an immutable document. The purpose of the subsequent analysis is to fully understand the problem and its implications. There are no reasons to expect that a problem statement prepared without a fully analysis will be correct.

The analyst must work with the requestor to refine the requirements so they represent the requestor's true intent. This involves challenging the requirements and probing for missing information. The psychological, organizational, and political considerations of doing this are beyond the scope of this book, except for the following piece of advice: If you do exactly what the customer asked for, but the result does not meet the customer's real needs, you will probably be blamed anyway.

Requirement Analysis

Functional Requirements:

By conducting the requirements analysis, we listed out the requirements that are useful to restate the problem definition.

- Insert the image into database
- Split the image into no of parts.
- Merge the parts.
- Identify the image.
- Draw image manually.
- Maintain information about each person

Minimum Hardware Requirements:

Processor: Pentium III – 900 MHz

Hard Disk: 20 GB

RAM: 128 MB

Functionality and Design

A newly-emerging trend in facial recognition software uses a 3D model, which claims to provide more accuracy. Capturing a real-time 3D image of a person's facial surface, 3D facial recognition uses distinctive features of the face -- where rigid tissue and bone is most apparent, such as the curves of the eye socket, nose and chin -- to identify the subject. These areas are all unique and don't change over time.

Using depth and an axis of measurement that is not affected by lighting, 3D facial recognition can even be used in darkness and has the ability to recognize a subject at different view angles with the potential to recognize up to 90 degrees (a face in profile).

Using the 3D software, the system goes through a series of steps to verify the identity of an individual.

Detection

Acquiring an image can be accomplished by digitally scanning an existing photograph (2D) or by using a video image to acquire a live picture of a subject (3D).

Alignment

Once it detects a face, the system determines the head's position, size and pose. As stated earlier, the subject has the potential to be recognized up to 90 degrees, while with 2D, the head must be turned at least 35 degrees toward the camera.

Measurement

The system then measures the curves of the face on a sub-millimeter (or microwave) scale and creates a template.

Representation

The system translates the template into a unique code. This coding gives each template a set of numbers to represent the features on a subject's face.

Matching

If the image is 3D and the database contain 3D images, then matching will take place without any changes being made to the image. However, there is a challenge currently facing databases that are still in 2D images. 3D provides a live, moving variable subject being compared to a flat, stable image. New technology is addressing this challenge. When a 3D image is taken, different points (usually three) are identified. For example, the outside of the eye, the inside of the eye and the tip of the nose will be pulled out and measured. Once those measurements are in place, an **algorithm** (a step-by-step procedure) will be applied to the image to

convert it to a 2D image. After conversion, the software will then compare the image with the 2D images in the database to find a potential match.

Verification or Identification

In verification, an image is matched to only one image in the database (1:1). For example, an image taken of a subject may be matched to an image in the Department of Motor Vehicles database to verify the subject is who he says he is. If identification is the goal, then the image is compared to all images in the database resulting in a score for each potential match (1:N). In this instance, you may take an image and compare it to a database of mug shots.

Design

During analysis, the focus is on what needs to be done, independent of how it is done. During design, decisions are made about how the problem will be solved, first at high level, then at increasingly detailed levels.

System design is the first design stage in which the basic approach to solving the problem is selected. During system design, the overall structure and style are decided. The system architecture is the overall organization of the system into components called subsystems. The architecture provides the context in which more detailed decisions are made in later design stages. By making high level decisions that apply to the entire system, the system designer partitions the problem into subsystems so that further work can be done by several designers working independently on different subsystems.

The system designer must make the following decisions:

- Organize the system into subsystems.
- Identify the concurrency inherent in the problem.
- Allocate subsystems to processors and tasks.
- Choose an approach for management of data stores.
- Handle access to global resources.
- Choose the implementation of control in software.
- Handle boundary conditions.
- Set trade-off priorities.

Breaking the System into Subsystems:

The first step in system design is to divide the system into small number of components. Each major component of a system is called a sub system. Each subsystem encompasses aspects of the system that share some common property – similar functionality, the same physical location, or execution on the same kind of hardware.

A subsystem not an object nor a function but a package of classes, associations, operations, events, and constraints that are interrelated and that have a reasonably well-defined and small interface with other subsystems. A subsystem usually identified by the services it provides. A service is a group of related functions that share some common purpose such as I/O processing. A subsystem defines a coherent way of looking at one aspect of the problem.

Each subsystem has a well-defined interface to the rest of the system. The interface specifies the form of all interactions and the information flow across subsystem boundaries but does not specify how the sub system is implemented internally. Each subsystem then can be designed independently without affecting the others.

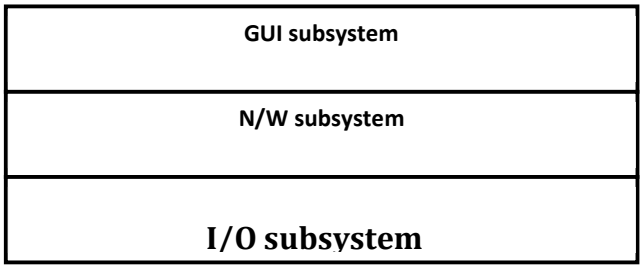
The decomposition of systems into subsystems may be organized as a sequence of horizontal layers or vertical partitions.

A layered system is an ordered set of virtual worlds, each built in terms of the ones below it and providing the basis of implementation for the ones above it.

The objects in each layer can be independent, although there is often some correspondence between objects in different layers. Knowledge is one-way only: a subsystem knows about the layers below it, but has no knowledge of the above layers. Each layer may have its own set of classes and operations. Each layer is implemented in terms of the classes and operations of lower layers.

Layered architecture comes in two forms: closed and open. In a closed architecture, each layer is built only in terms of the immediate lower layer. In an open architecture, a layer can use features of any lower layer to any depth.

We decomposed our system into three subsystems as layers. The three layers are closed architecture form. The three layers are GUI layer, Network layer and I/O layer. The purpose of GUI layer is to provide an efficient user interface to the user to interact with the system. It is built upon the Network layer which provides basic FTP services. The lowest layer is the I/O layer that provides services like reading or writing file to and from local and remote systems.



When top-level subsystems are identified, the designer should show the information flow among the sub systems. There are several architectural frameworks that are common in existing systems. They are batch transformation, continuous transformation, interactive interface, dynamic simulation, real-time system and transaction manager.

In the architectural frameworks specified above, our system will best suit in interactive interface architecture, since there are large number of interactions between system and user.

An interactive interface is a system that is dominated by interactions between the system and external agents, such as humans, devices or other programs. The external agents are independent of system, so their inputs can't be controlled, although the system may solicit responses from them.

Identifying Concurrency:

One important goal of system design is to identify which objects must be active concurrently and which objects have activity that is mutually exclusive. The latter objects can be folded together in a single thread of control or task. But there is no part that is concurrent in our system.

Allocating Subsystems to Processor:

In this step system designer estimates the hardware resources required and the implementation choice of either hardware or software. In our system all the

subsystems will be implemented in software. The hardware requirements are general such as Pentium – III, 128 MB of RAM.

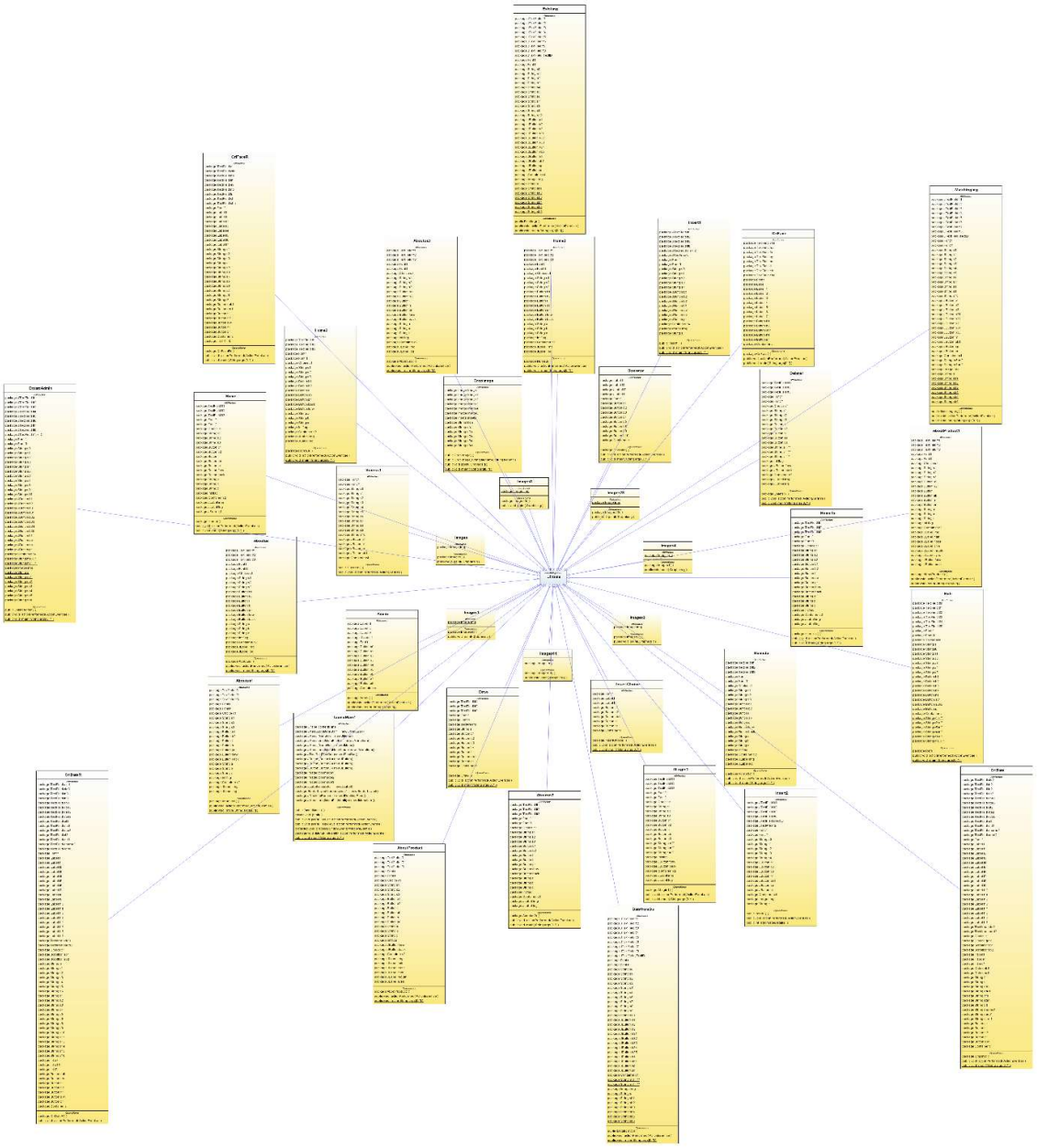
Management of Data Stores:

In this stage the system designer decides what format is used to store the data stores. There are DBMS systems or file systems and others. Here in our project there are no data stores except files. We then definitely prefer files to download and upload.

Choosing Software Control Implementation:

During the analysis, all interactions are shown as events between objects. But the system designer must choose among several ways to implement control in software. There are two kinds of control flows in a software system: internal and external. External control is the flow of externally visible events among the objects in the system. There are three kinds of control for external events: procedure driven, event driven sequential and concurrent. Internal control is the flow of control within a process.

Class Diagram:



SDLC Methodologies

This document plays a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, "A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

- The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
- A preliminary design is created for the new system.
- A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
- A second prototype is evolved by a fourfold procedure:

    1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
    2. Defining the requirements of the second prototype.

3. Planning an designing the second prototype.

4. Constructing and testing the second prototype.

- At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer's judgment, result in a less-than-satisfactory final product.
- The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
- The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
- The final system is constructed, based on the refined prototype.
- The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

The following diagram shows how a spiral model acts like:

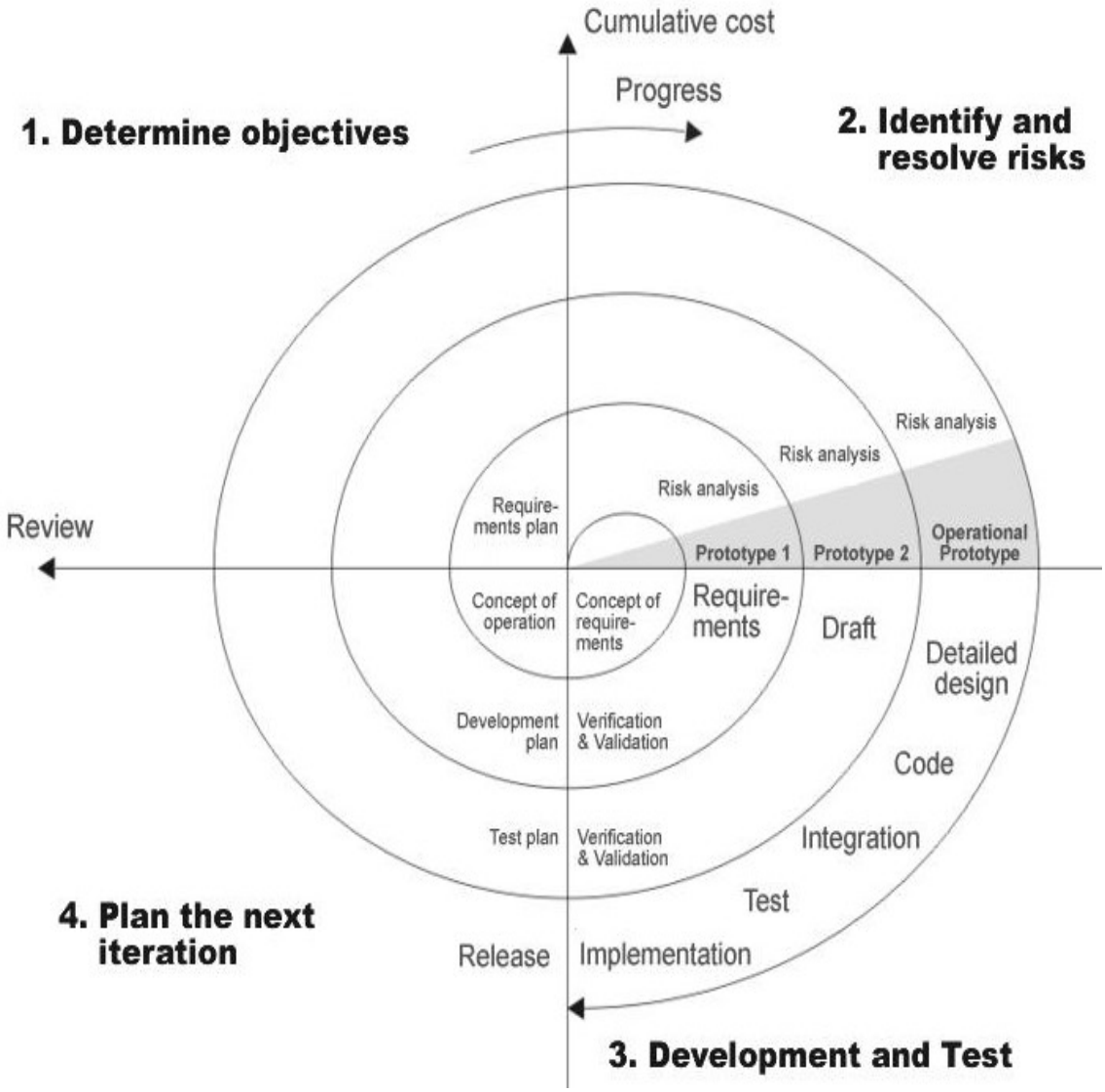


Fig 1.0-Spiral Model

ADVANTAGES:

- Estimates(i.e. budget, schedule etc .) become more relistic as work progresses, because important issues discoved earlier .
- It is more able to cope with the changes that are software development generally entails.
- Software engineers can get their hands in and start woring on the core of a project earlier.

Application Development

N-TIER APPLICATIONS

N-Tier Applications can easily implement the concepts of Distributed Application Design and Architecture. The N-Tier Applications provide strategic benefits to Enterprise Solutions. While 2-tier, client-server can help us create quick and easy solutions and may be used for Rapid Prototyping, they can easily become a maintenance and security night mare

The N-tier Applications provide specific advantages that are vital to the business continuity of the enterprise. Typical features of a real-life n-tier may include the following:

- Security
- Availability and Scalability
- Manageability
- Easy Maintenance
- Data Abstraction

The above-mentioned points are some of the key design goals of a successful n-tier application that intends to provide a good Business Solution.

DEFINITION

Simply stated, an n-tier application helps us distribute the overall functionality into various tiers or layers:

- Presentation Layer
- Business Rules Layer
- Data Access Layer
- Database/Data Store

Each layer can be developed independently of the other provided that it adheres to the standards and communicates with the other layers as per the specifications. This is the one of the biggest advantages of the n-tier application. Each layer can potentially treat the other layer as a 'Block-Box'.

In other words, each layer does not care how other layer processes the data as long as it sends the right data in a correct format.

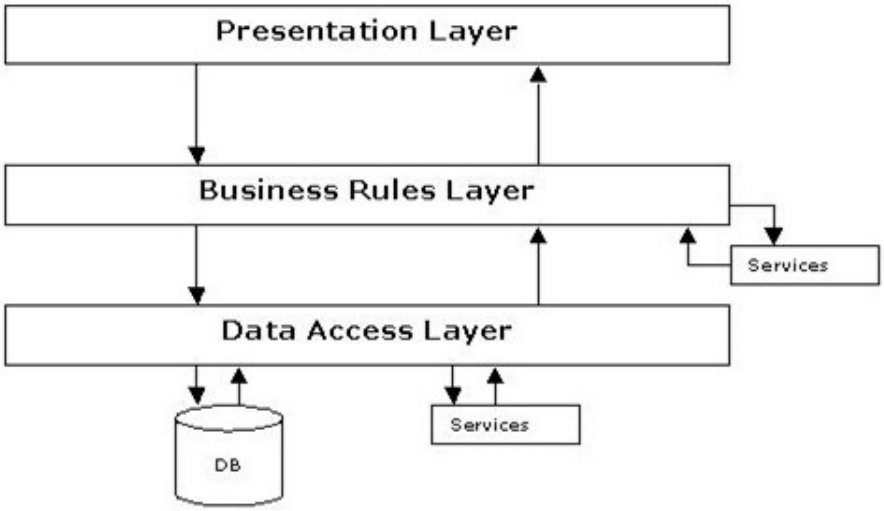


Fig 1.1-N-Tier Architecture

1. **THE PRESENTATION LAYER**

   Also called as the client layer comprises of components that are dedicated to presenting the data to the user. For example: Windows/Web Forms and buttons, edit boxes, Text boxes, labels, grids, etc.

2. **THE BUSINESS RULES LAYER**

   This layer encapsulates the Business rules or the business logic of the encapsulations. To have a separate layer for business logic is of a great advantage. This is because any changes in Business Rules can be easily handled in this layer. As long as the interface between the layers remains

the same, any changes to the functionality/processing logic in this layer can be made without impacting the others. A lot of client-server apps failed to implement successfully as changing the business logic was a painful process.

3. **THE DATA ACCESS LAYER**

This layer comprises of components that help in accessing the Database. If used in the right way, this layer provides a level of abstraction for the database structures. Simply put changes made to the database, tables, etc do not affect the rest of the application because of the Data Access layer. The different application layers send the data requests to this layer and receive the response from this layer.

4. **THE DATABASE LAYER**

This layer comprises of the Database Components such as DB Files, Tables, Views, etc. The Actual database could be created using SQL Server, Oracle, Flat files, etc.
In an n-tier application, the entire application can be implemented in such a way that it is independent of the actual Database. For instance, you could change the Database Location with minimal changes to Data Access Layer. The rest of the Application should remain unaffected.

Software Testing

Software testing is a critical element of software quality assurance and represents the ultimate reviews of specification, design and coding. Testing represents an interesting anomaly for the software. During earlier definition and development phases, it was attempted to build software from an abstract concept to a tangible implementation. No system is error free because it is so till the next error crops up during any phase of the development or usage of the product. A sincere effort however needs to be put to bring out a product that is satisfactory.

The testing phase involves the testing of development system using various data. Preparation of the test data plays a vital role in system testing. After preparing the test data, the system under study was tested using those data. While testing the system, by using the test data, errors were found and corrected by using the following testing steps and corrections were also noted for future use. Thus, a series of testing is performed on the proposed system before the system is ready for implementation.

The various types of testing done on the system are:

- Integration testing
- Validation testing
- Unit testing
- Output testing
- User Acceptance testing

Unit testing:

Unit testing focuses on verification effort on the smallest unit of software design module. Using the unit test plans prepared in the design phase of the system development as a guide, important control paths are tested to uncover errors with in the boundary of the modules. The interfaces of the modules are tested to ensure proper flow of information into and out of the modules under consideration boundary conditions were checked. All independent paths were exercised to ensure that all statements in the module have been executed at least once and all error-handling paths were tested.

Each unit is thoroughly tested to check if it might fail in any possible situation. This testing is carried during the programming state itself. At the end of this testing phase each module is found to be have an adverse effect working satisfactorily, as regard to the expected output from the module.

Integration Testing:

Data can be lost across an interface, one module can on another; sub-functions when combined may not produce the desired major function: global data structures can present problems. Integration testing is a systematic technique for the program structure while at the same time concluding tests to uncover errors associated with interface. All modules are combined in this testing step. Then the entire program is tested as a whole. Each of the module is integrated and

tested separately and later all modules are tested together for some time to ensure the system as a whole works well without any errors.

Validation Testing:

At the culmination of the integration testing, the software is completely assembled as a package, interfacing errors have been uncovered and corrected, and a final series of software validation testing began. Here we test if the system functions in a manner that can be reasonably expected by the customer. The system is tested against the system requirement specification.

Output Testing:

After performing validation testing, the next phase is output testing of the proposed system, since no system can be useful if it does not produce the desired output in the specified format. The output generated or displayed by the system under consideration is tested by asking the user about the format required by them, here, the output format is considered in two ways: One is on the screen and the other is on the printed form. Beta testing is carried output by the client, and minor errors that have been discovered by the client are rectified to improve the user friendliness of the system.

Object-Oriented Testing:

The overall objectives of the object-oriented testing – to find the maximum number of errors with a minimum amount effort – is identical to the to the objective of conventional software testing. But the strategy and tactics for OO testing differ significantly. The view of testing broadens to include the review of both the analysis and design model. In addition, the focus of testing moves away from the procedural component and toward the class.

Because the OO analysis and design models and the resulting source code are semantically coupled, testing begins during these engineering activities. For this reason, a review of CRC, object relationships, and object behaviour models can be viewed as first stage testing. As a result of this first stage testing, we encountered few problems in OOA done at analysis time. We have gone back and remodelled with new errorless classes and their relationships. The documented model is the revised model of earlier analysis model.

Once OOP has been accomplished, unit testing is applied for each class. Class testing uses a variety of methods: fault-based, random, and partition test methods. Each of those methods exercises the operations encapsulated by the class. Test sequences are designed to ensure that relevant operations are exercised. The state of the class, represented by the values of its attributes, is examined to determine if errors exist.

Integration test can be accomplished using a thread-based or use-based strategy. Thread-based strategy integrates the set of classes that collaborate to respond to one input or event. Use-based testing constructs the system in layers, beginning with those classes that do not make use of server classes. Integration test case design methods can also make use of random and partition tests. In addition, scenario-based testing and the tests derived from behavioural models can be used to test a class and its collaborators. A test sequence tracks the flow of operations across class collaborations.

OO system validation testing is black box oriented and can be accomplished by applying the same black box methods known for conventional software. However, scenario-based testing dominates the validation of OO systems, making the use case a primarily driver for validation testing.

Conclusion: This Project Report gives us a brief idea about how the Face Recognition actually works and how it can be used for a lot of purposes. Working on the project was good experience. I understand the importance of Planning and designing as a part of software development. But it's very difficult to complete the program for single person. Developing the project has given me some experience on real-time development Procedures.

References:

[1] Proyecto Fin de Carrera "Face Recognition Algorithms", June 16, 2010 [2] Peter N. Belhumeur, " Ongoing Challenges in Face Recognition".

[2] Kyong, K. and Jung, K.," GPU Implementation of Neural Network. Pattern Recognition", vol. 37, no. 6, pp. 1311-1314. Pergamon, 2004.

[3] Ekman,"Facial action Coding System",COmsulting Psycologistic Press,CA 1998.

[4] D. J. Field, "Relations between the statistics of natural images and the response properties of cortical cells," J. Opt. Soc. Amer. A, vol. 4, no. 12,pp. 2379–2394, Dec. 1987.

[5] Seyed Medi Lajevadi,Member IEEE,"Facial Expression Recognition in Perceptual Color Space" ,IEEE transaction on IMAGE PROCESSING,Vol.21,No.8,Aug.2012