CHAPTER-6

SOFTWARE ENGINEERING APPROACH

6.1 SOFTWARE ENGINEERING PARADIGM APPLIED:

6.1.1 DESCRIPTION:

INCREMENTAL MODEL

Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle. Incremental development is done in steps from analysis design, implementation, testing/verification, maintenance.

Each iteration passes through the **requirements, design, coding and testing phases**. And each subsequent release of the system adds function to the previous release until all designed functionality has been implemented.

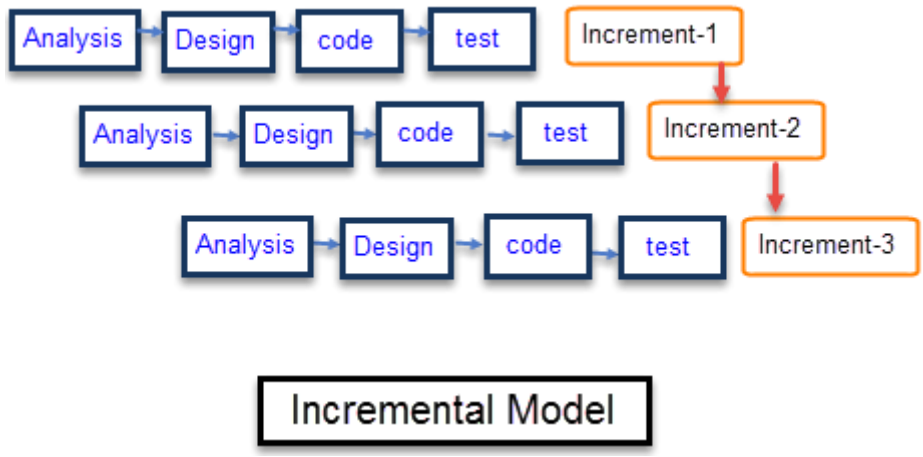


Fig 5.1 Incremental Model

The incremental build model is a method of software development where the model is designed, implemented and tested incrementally (a little more is added each time) until the

product is finished. It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements.

Characteristics of an Incremental model includes:

- System development is broken down into many mini development projects
- Partial systems are successively built to produce a final total system
- Highest priority requirement is tackled first
- Once the requirement is developed, requirement for that increment are frozen

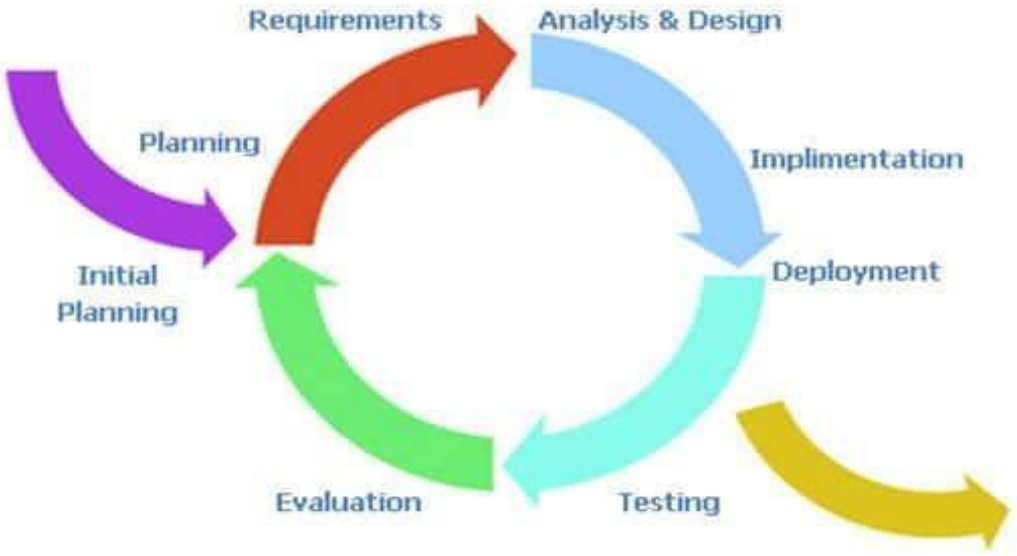


Fig 5.2 Characteristics

When to use Incremental models?

- Requirements of the system are clearly understood • When demand for an early release of a product arises.
- When high-risk features and goals are involved.

- Such methodology is more in use for web application and product based companies.
- When Projects having lengthy developments schedules.
- A new technology is being used.

6.1.2 Advantages And Disadvantages:

Advantages of Incremental Model

- Generates working software quickly and early during the software life cycle.
- More flexible – less costly to change scope and requirements.
- Easier to test and debug during a smaller iteration.
- Easier to manage risk because risky pieces are identified and handled during its iteration.
- Each iteration is an easily managed milestone.
- Thought the development stages changes can be done

Disadvantages of Incremental Model

- It requires a good planning designing
- Each phase of an iteration is rigid and do not overlap each other.
- Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.
- Well defined module interfaces are required.

6.1.3 Reason For Use:

As we know all the requirements of our project and are clearly understood.

We have build our project in different components, Like:-

Component 1. Enter Enrollnment No & Name.

Component 2. Access Webcam for capturing images.

Component 3. Train Machine Learning Model based on Recognized Images.

Component 4. Attendance is generated by ML Model, and can be edited manually.

6.2 REQUIREMENT ANALYSIS

6.2.1 SOFTWARE REQUIREMENT SPECIFICATION

Software requirement specification (SRS) is a technical specification of requirements for the software product. SRS represents an overview of products, features and summaries the processing environments for development operation and maintenance of the product. The goal of the requirement specification phase is to produce the software specification document also called requirement document.

Requirement Specification

This requirement specification must have system properties. Conceptually every SRS should have the components:

- Functionality
- Performance
- Design constraints imposed on an implementation
- External interfaces

6.2.1.1 GLOSSARY

- **Python**: It is an interpreted, high-level, general-purpose programming language. It is used for:

○ web development (server-side)

○ software development

○ mathematics

○ system scripting

● **OpenCV**: OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. It was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

● **Numpy**: NumPy is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

● **Keras**:Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

● **Tensorflow**:TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks

6.2.2.2 SUPPLEMENTARY SPECIFICATIONS

Functional Requirements:

Since AASFR consists of three major parts named Face Detection (using OpenCV), Face Recognition (using Machine Learning Model) and Attendance Generation (both Automated and Manual), functional requirements will be examined in three chapters:

- Functional Requirements of Face Detection (using OpenCV)
- Functional Requirements of Face Recognition (using Machine Learning Model)
- Functional Requirements of Attendance Generation (both Automated and Manual)

Functional Requirements of Face Detection (using OpenCV)

FR-01: Clear Field of View

There must not be an object between the laptop and hand of the user in order for a successful tracking.

FR-02 Face Detection Module

This software shall perform face detection and filter out all distorted faces (out of range). Thus now the system would detect . The object marker detection module can be achieved by using the detect contour method.

FR-03 Filtered Object Detection

Once the program has filtered out most of the unwanted parts of the picture after using the detect_contour module, the software shall read and recognize biggest objects also known as "markers".This allows the system to pinpoint possible locations of user's object markers.

FR-04 Object Location

After detection, the system shall be now be able to compute the location of the mean points(of each color), using simple numpy module.

FR-05: Close:

Cross Functionality has been added with users confirmation

Functional Requirements of Face Recognition:

FR-06 Detect faces:

The initial step of our pipeline includes detecting the face for which we will use Histogram of Oriented Gradients (HOG) . One of the most popular and successful "person detectors" out there right now is the HOG with SVM approach.HOG stands for Histograms of Oriented Gradients.

HOG is a type of "feature descriptor". The intent of a feature descriptor is to generalize the object in such a way that the same object (in this case a person) produces as close as possible to the same feature descriptor when viewed under different conditions. This makes the classification task easier.

FR-07Analyze facial features (Posing and Projecting Faces):
After differentiating the faces from the image we have to now deal with the problem of face turning into different directions which would look totally different to a computer.
Humans can easily recognize that both images are of the same person, but computer would see these pictures as two completely different people. To account for this, we will try to warp each picture so that the eyes and lips are always in the sample place in the image. This will make it a lot easier for us to compare face in the next steps. For this, we are going to use an algorithm called face landmark estimation.

FR-08Encoding Images:

Next step involves recognizing the detected face for which we need to extract a few basic measurements from each face. Then we could measure our unknown face the same way and find the known face with closest measurements. The most accurate approach is to let the computer figure out the measurements to collect itself. Deep learning does a better job than humans at figuring out which parts of a face are important to measure. The solution is to train a Deep Convolution Neural Network to generate 128 measurements for each face.

Functional Requirements of Attendance Generation:

FR-09 Generating Attendance:

After recognition of the faces the details of student like Enrollment no & Name is saved into the excel file.

FR-10 Maually marking the attendance:
Marking the attendance of those students whose attendance cannot be generated

Non-functional Requirements:

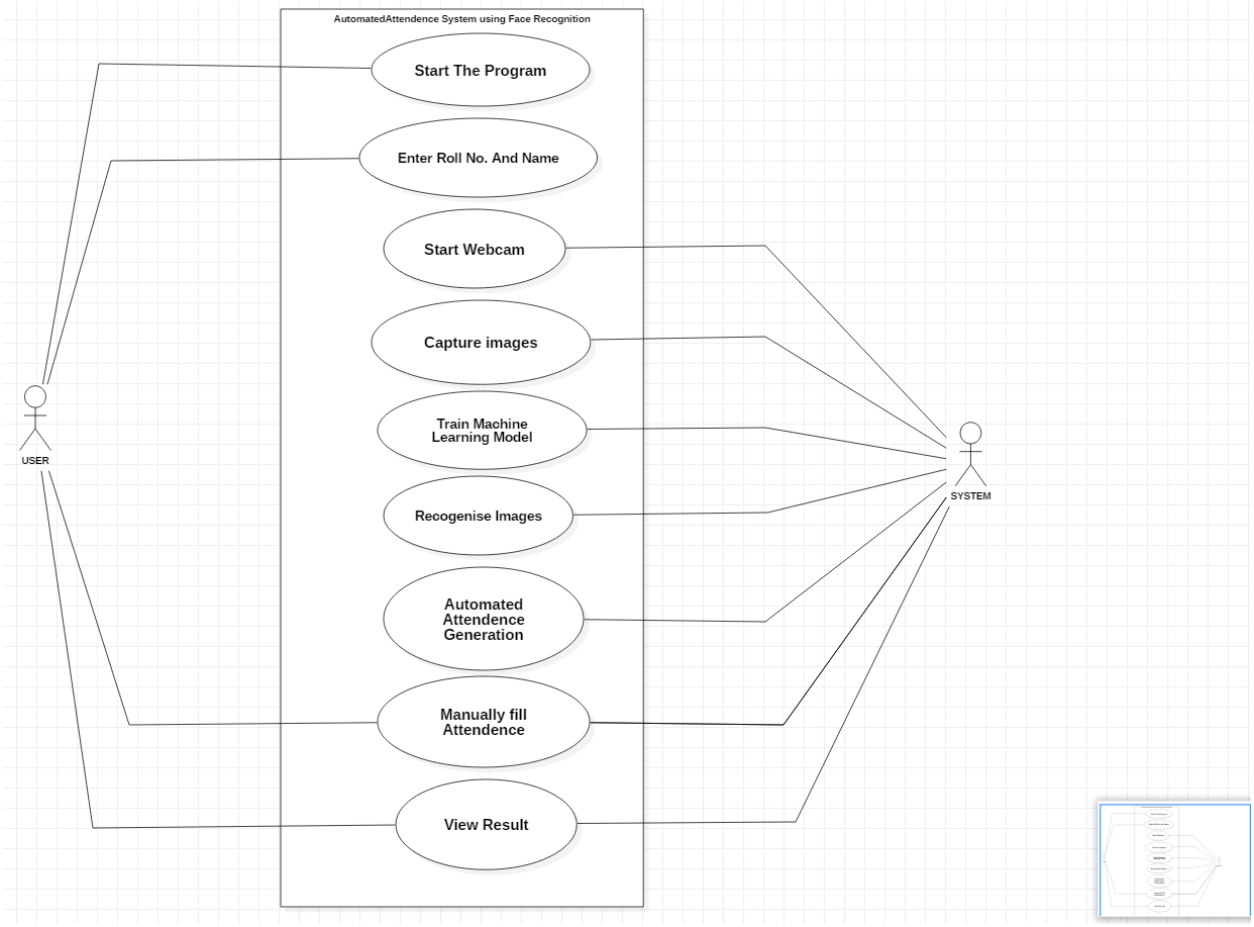
- **Usability:** The user is facilitated with the control section for the entire process in which they can arrange the position of marker at the FOV(field of view) under consideration, the variation of marker position and respective command

generation etc can be effectively facilitated by mean of user interface. The implementation and calibration of camera and its resolution can also be done as per quality and preciseness requirement. The frame size, flow rate and its command variation with respect to threshold developed and color component of marker color, can be easily calibrated by means of certain defined thresholds.

- **Security and support:** Application will be permissible to be used only in secure network so there is less feasibility of insecurity over the functionality of the application. On the other hand, the system functions in a real time application scenario, therefore the camera, color and platform compatibility is also must in this case.
- **Maintainability:** The installation and operation manual of the project will be provided to the user.
- **Extensibility:** The project work is also open for any future modification and hence the work could be defined as the one of the extensible work.

6.2.2.3 USE CASE MODEL

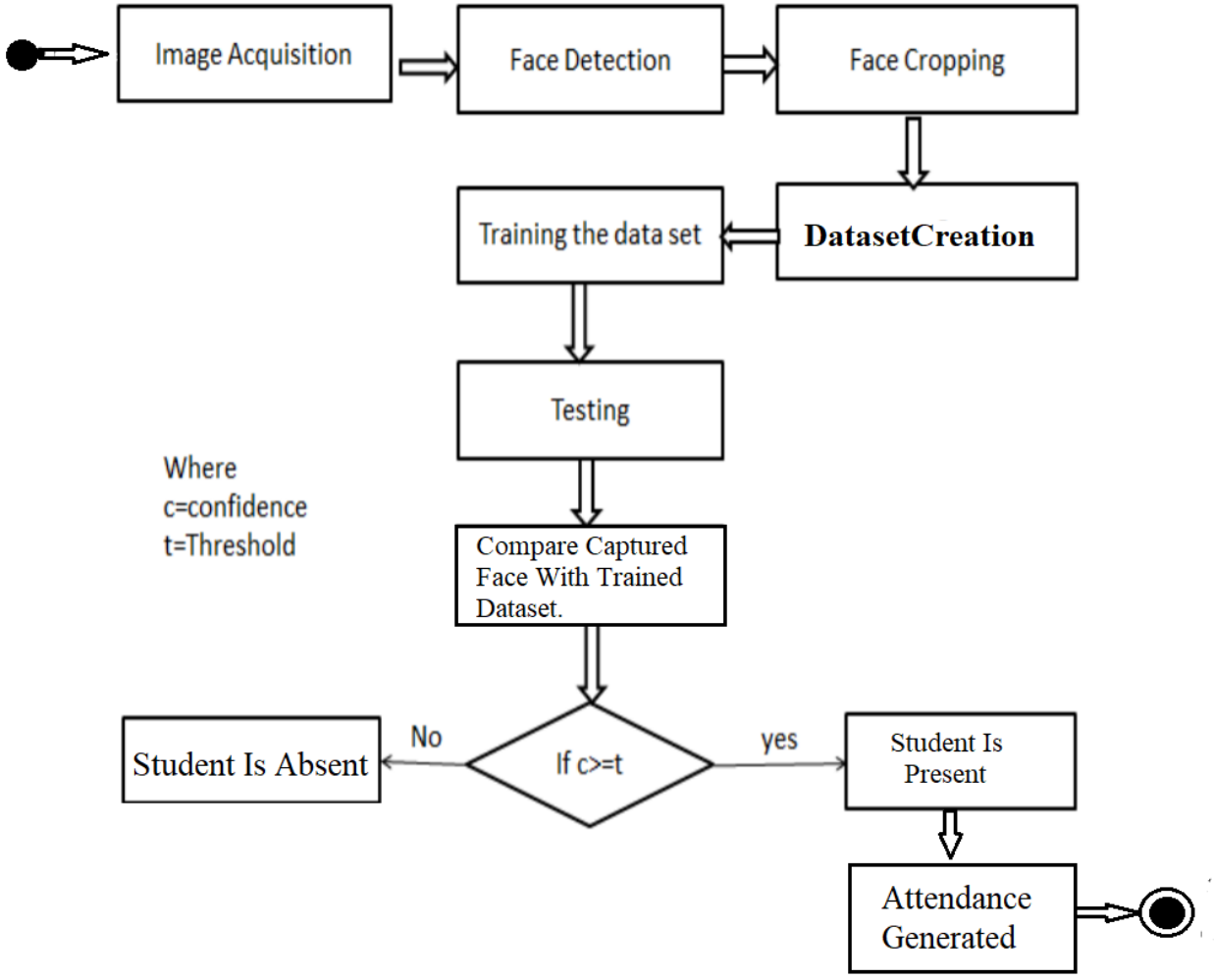
In software and systems engineering, a use case is a list of actions or event steps, typically defining the interactions between a role (known in the Unified Modelling Language as an actor) and a system, to achieve a goal. The actor can be a human or other external system. Use case analysis is an important and valuable requirement analysis technique that has been widely used in modern software engineering since its formal introduction by Ivar Jacobson in 1992. Use case driven development is a key characteristic of many process models and frameworks such as ICONIX, the Unified Process (UP), the IBM Rational Unified Process (RUP), and the Oracle Unified Method (OUM). With its inherent iterative, incremental and evolutionary nature, use case also fits well for agile development. The purpose of the use case diagram is to capture the dynamic aspect of a system. Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analysed to gather its functionalities use cases are prepared and actors are identified. Now when the initial task is complete use case diagrams are modelled to present the outside view.



6.2.2. CONCEPTUAL LEVEL ACTIVITY DIAGRAM

An activity diagram is a UML diagram that models the dynamic aspects of a system. It is a simplification of the UML state chart diagram for modeling control flows in computational and organizational processes. It allows you to represent a functional decomposition of system behaviour. An activity diagram provides a complete specification of behaviour and not, like the interaction diagrams, a single possible scenario.

The activity diagram gives a simplified representation of a process, showing control flows (called transitions) between actions performed in the system (called activities). These flows represent the internal behaviour of a model element (use case, package, classifier or operation) from a start point to several potential endpoints



6.3 PLANNING MANAGERIAL ISSUES

6.3.1 PLANNING SCOPE

The first activity in software project planning is the determination of software scope. Software scope describes the data and control to be processed, function, performance, constraints, interfaces, and reliability. Functions described in the statement of scope are evaluated and in some cases refined to provide more detail prior to the beginning of estimation. Because both cost and schedule estimates are functionally oriented, some degree of decomposition is often useful. Performance considerations encompass processing and response time requirements. Constraints identify limits placed on the software by external hardware, available memory, or other existing systems.

6.3.2 PROJECT RESOURCES

The second software planning task is estimation of the resources required to accomplish the software development effort.

The development environment—hardware and software tools—sits at the foundation of the resources pyramid and provides the infrastructure to support the development effort. At a higher level, we encounter reusable software components—software building blocks that can dramatically reduce development costs and accelerate delivery. At the top of the pyramid is the primary resource—people.

Each resource is specified with four characteristics: description of the resource, a statement of availability, time when the resource will be required; duration of time that resource will be applied. The last two characteristics can be viewed as a time window. Availability of the resource for a specified window must be established at the earliest practical time.

·Human Resources

The planner begins by evaluating scope and selecting the skills required to complete development. Both organizational positions (e.g., manager, senior software engineer) and specialty (e.g., telecommunications, database, and client/server) are specified. For relatively small projects (one person-year or less), a single individual may perform all software engineering tasks, consulting with specialists as required. The number of people required for a software project can be determined only after an estimate of development effort (e.g., person-months) is made.

Reusable Software Resources

Component-based software engineering (CBSE) emphasizes reusability—that is, the creation and reuse of software building blocks. Such building blocks, often called components, must be cataloged for easy reference, standardized for easy application, and validated for easy integration.

- **Off-the-shelf components**: Existing software that can be acquired from a third party or that has been developed internally for a past project.
- **Full-experience components**: Existing specifications, designs, code, or test data developed for past projects that are similar to the software to be built for the current project. Members of the current software team have had full experience in the application area represented by these components. Therefore, modifications required for full-experience components will be relatively low-risk.
- **Partial-experience components**: Existing specifications, designs, code, or test data developed for past projects that are related to the software to be built for the current project but will require substantial modification. Members of the current software team have only limited experience in the application area represented by

these components. Therefore, modifications required for partial-experience components have a fair degree of risk.

- **New components:** Software components that must be built by the software team specifically for the needs of the current project.

.

Environmental Resources

The environment that supports the software project, often called the software engineering environment (SEE), incorporates hardware and software. Hardware provides a platform that supports the tools (software) required to produce the work products that are an outcome of good software engineering practice. Because most software organizations have multiple constituencies that require access to the SEE, a project planner must prescribe the time window required for hardware and software and verify that these resources will be available.

When a computer-based system (incorporating specialized hardware and software) is to be engineered, the software team may require access to hardware elements being developed by other engineering teams. For example, software for a numerical control (NC) used on a class of machine tools may require a specific machine tool (e.g., an NC lathe) as part of the validation test step; a software project for advanced page-layout may need a digital-typesetting system at some point during development. Each hardware element must be specified by the software project planner.

6.3.3 TEAM ORGANIZATION:

The following options are available for applying human resources to a project that will require n people working for k years:

1. N individuals are assigned to M different functional tasks, relatively little combined work occurs; coordination is the responsibility of a software manager who may have six other projects to be concerned with.
2. N individuals are assigned to m different functional tasks (m< n) so that informal "teams" are established; an ad hoc team leader may be appointed; coordination among teams is the responsibility of a software manager.
3. N individuals are organized into t teams; each team is assigned one or more functional tasks; each team has a specific structure that is defined for all teams working on a project; coordination is controlled by both the team and a software project manager.

Although it is possible to voice arguments for and against each of these approaches, a growing body of evidence indicates that a formal team organization (option 3) is most productive.

The "best" team structure depends on the management style of your organization, the number of people who will populate the team and their skill levels, and the overall problem difficulty.

How should a software team be organized?

Democratic decentralized (DD): This software engineering team has no permanent leader. Rather, "task coordinators are appointed for short durations and then replaced by others who may coordinate different tasks." Decisions on problems and approach are made by group consensus. Communication among team members is horizontal.

Controlled decentralized (CD): This software engineering team has a defined leader who coordinates specific tasks and secondary leaders that have responsibility for subtasks. Problem solving remains a group activity, but implementation of solutions is partitioned among subgroups by the team leader. Communication among subgroups and individuals is horizontal. Vertical communication along the control hierarchy also occurs.

Controlled Centralized (CC):Top-level problem solving and internal team coordination are managed by a team leader. Communication between the leader and team members is vertical.

Seven project factors that should be considered when planning the structure of software engineering teams:

- The difficulty of the problem to be solved.
- The size of the resultant program(s) in lines of code or function points • The time that the team will stay together (team lifetime).
- The degree to which the problem can be modularized.
- The required quality and reliability of the system to be built.
- The rigidity of the delivery date.
- The degree of sociability (communication) required for the project.

Because a centralized structure completes tasks faster, it is the most adept at handling simple problems. Decentralized teams generate more and better solutions than individuals. Therefore such teams have a greater probability of success when working on difficult problems. Since the CD team is centralized for problem solving, either a CD or CC team structure can be successfully applied to simple problems. A DD structure is best for difficult problems.

Because the performance of a team is inversely proportional to the amount of communication that must be conducted, very large projects are best addressed by team with a CC or CD structures when subgrouping can be easily accommodated. It has been found that DD team structures result in high morale and job satisfaction and are therefore good for teams that will be together for a long time. The DD team structure is best applied to problems with relatively low modularity, because of the higher volume of communication needed. When high modularity is possible (and people can do their own thing), the CC or CD structure will work well.

It's often better to have a few small, well-focused teams than a single large team. CC and CD teams have been found to produce fewer defects than DD teams, but these data have much to do

with the specific quality assurance activities that are applied by the team. Decentralized teams generally require more time to complete a project than a centralized structure and at the same time are best when high sociability is required.

6.3.4 PROJECT SCHEDULING

Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks.
It is important to note, however, that the schedule evolves over time. During early stages of project planning, a macroscopic schedule is developed. This type of schedule identifies all major software engineering activities and the product functions to which they are applied. As the project gets under way, each entry on the macroscopic schedule is refined into a detailed schedule. Here, specific software tasks (required to accomplish an activity) are identified and scheduled. Scheduling for software engineering projects can be viewed from two rather different perspectives. In the first, an end-date for release of a computer-based system has already (and irrevocably) been established. The software organization is constrained to distribute effort within the prescribed time frame. The second view of software scheduling assumes that rough chronological bounds have been discussed but that the end-date is set by the software engineering organization. Effort is distributed to make best use of resources and an end-date is defined after careful analysis of the software. Unfortunately, the first situation is encountered far more frequently than the second.

Basic principles for software project scheduling:

- **Compartmentalization**: The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and the process are decomposed
- **Interdependency**: The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence while others can occur in parallel. Some activities cannot commence until the work product produced by another is available. Other activities can occur independently.
- **Time allocation**: Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort). In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis.
- **Effort validation**: Every project has a defined number of staff members. As time allocation occurs, the project manager must ensure that no more than the allocated number of people has been scheduled at any given time.

- **Defined responsibilities:** Every task that is scheduled should be assigned to a specific team member.
- **Defined outcomes**: Every task that is scheduled should have a defined outcome. For software projects, the outcome is normally a work product (e.g., the design of a module) or a part of a work product. Work products are often combined in deliverables.
- **Defined milestones**: Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality and has been approved.

6.3.5 ESTIMATION

● The accuracy of a software project estimate is predicated on:

○ The degree to which the planner has properly estimated the size (e.g., KLOC) of the product to be built

○ The ability to translate the size estimate into human effort, calendar time, and money

○ The degree to which the project plan reflects the abilities of the software team

○ The stability of both the product requirements and the environment that supports the software engineering effort

PROJECT ESTIMATION OPTIONS

● Options for achieving reliable cost and effort estimates

○ Delay estimation until late in the project (we should be able to achieve 100% accurate estimates after the project is complete)

○ Base estimates on similar projects that have already been completed

○ Use relatively simple decomposition techniques to generate project cost and effort estimates

○ Use one or more empirical estimation models for software cost and effort estimation.

Option #1 is not practical, but results in good numbers.

Option #2 can work reasonably well, but it also relies on other project influences being roughly equivalent

Options #3 and #4 can be done in tandem to cross check each other.

PROJECT ESTIMATION APPROACHES

● Decomposition techniques
  - o These take a "divide and conquer" approach
  - o Cost and effort estimation are performed in a stepwise fashion by breaking down a project into major functions and related software engineering activities

● Empirical estimation models

o Offer a potentially valuable estimation approach if the historical data used to seed the estimate is good

6.3.6 RISK ANALYSIS

Risk analysis and management are a series of steps that help a software team to understand and manage uncertainty. Many problems can plague a software project. There is general agreement that risk always involves two characteristics: **Uncertainty**—the risk may or may not happen; that is, there are no 100% probable risks.

Loss—if the risk becomes a reality, unwanted consequences or losses will occur.

When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk. To accomplish this, different categories of risks are considered.

- **Project risks**: if project risks become real, it is likely that project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, customer, and requirements problems and their impact on a software project.
- **Technical risks**: It threatens the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible. Technical risks identify potential design, implementation, interface, verification, and maintenance problems. Technical risks occur because the problem is harder to solve than we thought it would be.
- **Business risks**: It threatens the viability of the software to be built. Business risks often jeopardize the project or the product
- **Known risks** are those that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date, lack of documented requirements or software scope, poor development environment).
- **Predictable risks** are extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced).
- **Unpredictable risks** are the joker in the deck. They can and do occur, but they are extremely difficult to identify in advance.

There are a few well-known types of risk analysis that can be used. In software engineering, risk analysis is used to identify the high-risk elements of a project. It provides ways of documenting the impact of risk mitigation strategies. Risk analysis has also been shown to be important in the software design phase to evaluate criticality of the system, where risks are analyzed and necessary countermeasures are introduced . The purpose of risk analysis is to understand risk

better and to verify and correct attributes. A successful analysis includes essential elements like problem definition, problem formulation, data collection.

Risk Tree Analysis and Assessment Method:

In risk tree analysis method, software risks are classified at first. Then risks are identified in each group. Afterwards, primary or basic risk events, intermediate events, top event, and the necessary sub-tree are found. All these require that managers have a complete knowledge about the projects. Then the risk tree can be constructed. Likelihood and impact must be assigned to each event and failure.

Then probabilities starting from primary events to the top event are calculated. The events are ordered according to their probabilities. Maximum probability indicates the importance of those events; therefore, it is necessary to attend more to them. Managers should use solutions to prevent risks from occurring or reduce undesirable incidents.

The presented classifications and risk tree structures can apply with some software tools. Fault Tree Creation and Analysis Program, Fault Tree Tool or Relax Fault Tree can be used for this analysis. These tools have facilities that help users to create tree symbols and construct the risk tree structures.

6.3.7 SECURITY PLAN

Security plan include these steps:

1. Identify the assets you want to protect and the value of these assets.
2. Identify the risks to each asset.
3. Determine the category of the cause of the risk (natural disaster risk, intentional risk, or unintentional risk).
4. Identify the methods, tools, or techniques the threats use.

After assessing your risk, the next step is proactive planning. Proactive planning involves developing security policies and controls and implementing tools and techniques to aid in security.

The various types of policies that could be included are:

- Password policies

○ Administrative Responsibilities

○ User Responsibilities

- E-mail policies
- Internet policies
- Backup and restore policies

6.4. DESIGN

6.4.1 DESIGN CONCEPT

The purpose of the design phase is to plan a solution to the problem specified by the requirement of the problem specified by the requirement document. This phase is the first step in moving from the problem domain to the solution domain. In other words, starting with what is needed, the design takes us towards how to satisfy our needs. The design of the system is the most critical factor affecting the quality of the software and has a major impact on testing and maintenance. The output of this phase is the design document.

6.4.2. DESIGN TECHNIQUE

The design techniques that take place are:

- Stepwise Refinement
- Levels of abstraction
- Structured Design
- Integrated Top-Down Development
- Jackson Structured Programming

Stepwise Refinement: Stepwise refinement is a top-down technique for decomposing a system from the high-level specification into more elementary levels. It decomposes design decisions to elementary levels and isolating the design aspects that are not independent. It postpones decisions concerning representation details. Then it carefully demonstrates that each successive step in the refinement process is a faithful expansion of previous steps.

Levels of abstraction: It is a bottom-up design technique in which an operating system was designed as a layering of hierarchical levels starting at level 0. Internal functions are hidden from other levels, that can only be invoked by the functions on the same level. Each level of abstraction performs a set of services for the functions on the next higher level of abstraction. Higher level functions can invoke lower level functions but lower level functions cannot invoke higher level functions.

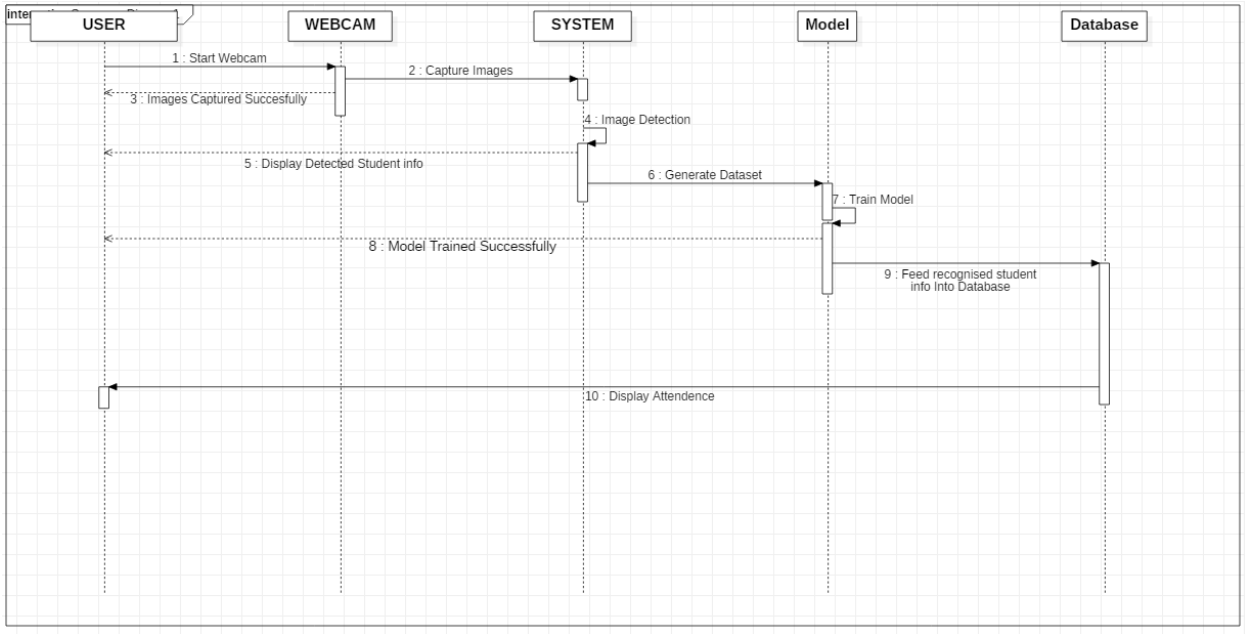
Structured Design: It is a top-down approach for the architectural design of software systems. It converts data flow diagrams into structure charts. Its first step is to review and refine the data flow diagrams developed during requirements definition and external design. It works on divide and conquer strategy.

Integrated top-down development: It integrates design, implementation and testing. Using integrated top-down development, design proceeds top-down from the highest-level routines; Lower-level routines may be implementations of an elementary function. There is thus a hierarchical structure to a top-down system.

Jackson Structured Programming: is a method for structured programming based on correspondences between data stream structure and program structure. JSP structures programs and data in terms of sequences, iterations and selections, and as a consequence it is applied when designing a program's detailed control structure. The method applies to processing of any data structure or data stream that is describable as a hierarchical structure of sequential, optional and iterated elements.

6.4.3. MODELING

6.4.3.1. SEQUENCE DIAGRAM



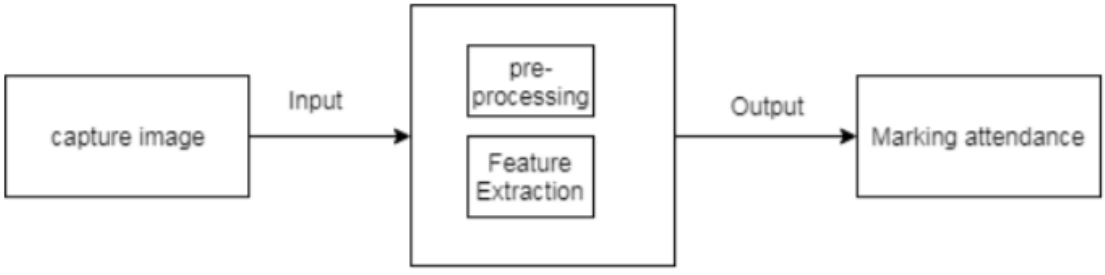
6.4.3.2. DATA FLOW MODEL

A data flow model is diagramatic representation of the flow and exchange of information within a system. Data flow models are used to graphically represent the flow of data in an information system by describing the processes involved in transferring data from input to file storage and reports generation.

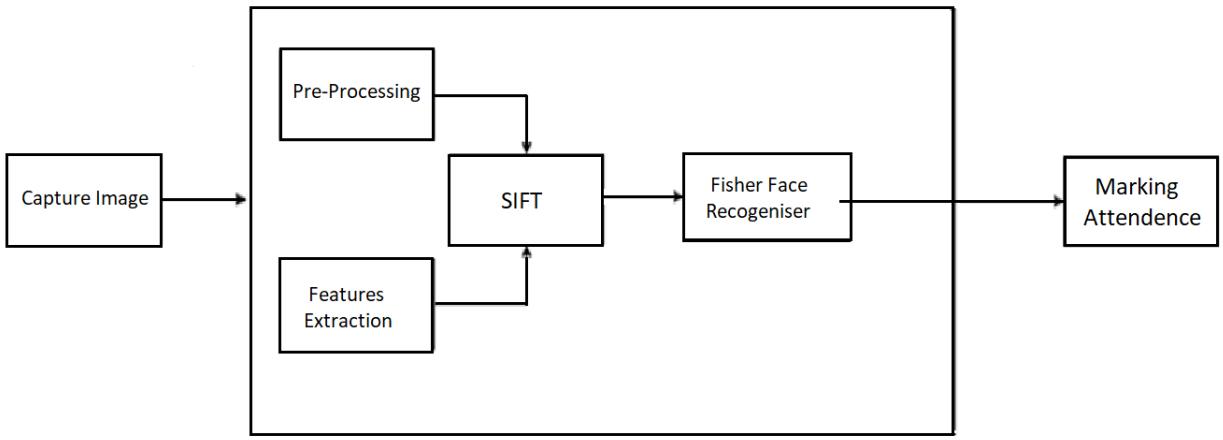
Data flow modeling can be used to identify a variety of different things, such as:

- Information that is received from or sent to other individuals, organizations, or other computer systems
- Areas within a system where information is stored and the flows of information within the system are being modeled
- The processes of a system that act upon information received and produce the resulting outputs

Level 1: DFD



Level 2: DFD



Level 3: DFD

6.5 IMPLEMENTATION PHASE

6.5.1 LANGUAGE USED CHARACTERISTICS

The language used to build this project is “Python”.

Python is a clear and powerful object-oriented programming language, comparable to Perl, Ruby, Scheme, or Java.

Some of the major characteristics of this language are as follows:

1. Easy

When we say the word ‘easy’, we mean it in different contexts.

2.Easy to code

As we have seen in earlier lessons, Python is very easy to code. Compared to other popular languages like Java and C++, it is easier to code in Python. Anyone can learn python syntax in just a few hours. Though sure, mastering Python requires learning about all its advanced concepts and packages and modules. That takes time. Thus, it is programmer-friendly.

3.Easy to read

Being a high-level language, Python code is quite like English. Looking at it, you can tell what the code is supposed to do. Also, since it is dynamically-typed, it mandates indentation. This aids readability.

4.Expressive

First, let's learn about expressiveness. Suppose we have two languages A and B, and all programs that can be made in A can be made in B using local transformations. However, there are some programs that can be made in B, but not in A, using local transformations. Then, B is said to be more expressive than A. Python provides us with a myriad of constructs that help us focus on the solution rather than on the syntax. This is one of the outstanding python features that tells you why you should learn Python.

5Free and Open-Source

Firstly, Python is freely available.

Secondly, it is open-source. This means that its source code is available to the public. You can download it, change it, use it, and distribute it. This is called FLOSS(Free/Libre and Open Source Software). As the Python community, we're all headed toward one goal- an ever-bettering Python .

6 High- Level

As we discussed in point 2b, it is a high-level language. This means that as programmers, we don't need to remember the system architecture. Nor do we need to manage the memory. This makes it more programmer-friendly and is 1 of the key python features.Any doubt yet in the features of Python**.**

7 Portable

Let's assume you've written a Python code for your Windows machine. Now, if you want to run it on a Mac, you don't need to make changes to it for the same. In other words, you can take one code and run it on any machine, there is no need to write different code for different machines. This makes Python a portable language. However, you must avoid any system-dependent features in this case.

8 Interpreted

In case of languages like C++ or Java, one must first compile it, and then run it. But in Python, there is no need to compile it. Internally, its source code is converted into an immediate form called bytecode. So, all you need to do is to run your Python code without worrying about linking to libraries, and a few other things.

By interpreted, we mean the source code is executed line by line, and not all at once. Because of this, it is easier to debug the code. Also, interpreting makes it just slightly slower than Java, but that does not matter compared to the benefits it has to offer.

9 Object-Oriented

A programming language that can model the real world is said to be object-oriented. It focuses on objects, and combines data and functions. Contrarily, a procedure-oriented language revolves

around functions, which are code that can be reused. Python supports both procedure-oriented and object-oriented programming which is one of the key python features. It also supports multiple inheritance, unlike Java. A class is a blueprint for such an object. It is an abstract data type, and holds no values.

10 Extensible

If needed, some of the Python code can be written in other languages like C++. This makes Python an extensible language, meaning that it can be extended to other languages.

11 Embeddable

Not only can we put code of other languages in our Python source code, it is also possible to put our Python code in a source code in a different language like C++. This allows us to integrate scripting capabilities into our program of the other language.

12 Large Standard Library

Python downloads with a large library that you can use so you don't have to write your own code for every single thing. There are libraries for regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and a lot of other functionality.

13 GUI Programming

One can use Tkinter module available to create basic GUIs.

14 Dynamically Typed

Python is dynamically-typed. This means that the type for a value is decided at runtime, not in advance. This is why we don't need to specify the type of data while declaring it.

6.5.2 Coding:

Python Files:

We have created various python files which are designed to do different functions and works. Graphical User Interface (GUI) has been developed using Tkinter in Python. Database used is MySQL in WAMPServer.

- **AMS_Run.py:**
  This is the main file where all the components of Front end & Back end are defined and used. It is used to run the Web Application, and consists of all the classes derived from various other files.
- **GUI_RS.py:**
  This file consists of all the GUI components, and the classes and functions required to run the Tkinter module.
- **Register_Student.py:**

This file is used to register the students in the database by faculty. Only these students are eligible for attendance.

- **Training.py:**
  This file is used to train the Machine learning model on the generated dataset, which is constructed by SVM and Fisher Face Recognizer.
- **Testing.py:**
  This file is used to test the generated dataset, and is also used to increase the accuracy of the trained model.

GUI_RS.py

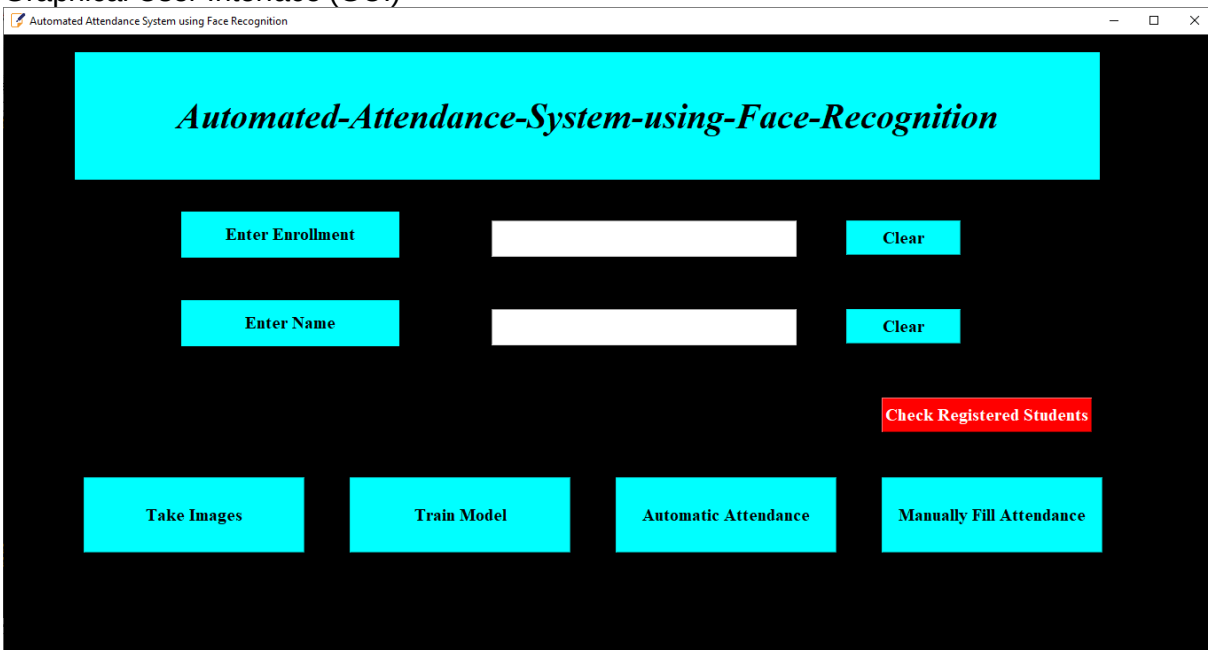
```
import tkinter as tk
from tkinter import Message, Text
import os
import shutil
import csv
import numpy as np
from PIL import Image,ImageTk
import cv2
import pandas as pd
import datetime
import time
import tkinter.ttk as ttk
import tkinter.font as font

window = tk.Tk()
window.title("Face_Recogniser")

dialog_title = 'QUIT'
dialog_text = 'Are you sure?'
window.geometry('1280x720')
window.configure(background='snow')

def clear():
    txt.delete(first=0, last=22)

def clear1():
    txt2.delete(first=0, last=22)

def take_img():
    try:
        cam = cv2.VideoCapture(0)
        detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
        stat = 'We are getting your face images.....'
        Notification.configure(text=stat, bg="SpringGreen2", width=23)
        Notification.place(x=350, y=400)
```

```
        Enrollment = txt.get()
        Name = txt2.get()
        path = 'C:/Users/kusha/PycharmProjects/Attendace managemnt system/Datasets/' +
Enrollment + ' ' + Name
        new_path = 'C:/Users/kusha/PycharmProjects/Attendace managemnt system/Datasets/' +
Enrollment + ' ' + Name + '/'
        os.mkdir(path)
        sampleNum = 0

        while (True):
            ret, img = cam.read()
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            faces = detector.detectMultiScale(gray, 1.3, 5)

            for (x, y, w, h) in faces:
                cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
                # incrementing sample number
                sampleNum = sampleNum + 1
                # saving the captured face in the dataset folder
                cv2.imwrite(new_path + str(sampleNum) + ".jpg", gray[y:y + h, x:x + w])
                cv2.imshow('frame', img)
                break
            # break if the sample number is morethan 100
            elif sampleNum > 150:
                break
        cam.release()
        cv2.destroyAllWindows()
        res = "Images Saved for Enrollment : " + Enrollment + " Name : " + Name
        Notification.configure(text=res, bg="SpringGreen3", width=50,font=('times',18, 'bold'))
        Notification.place(x=350, y=400)

    except FileExistsError as F:
        f='Student Data already exists'
        Notification.configure(text=f, bg="Red",width=21)
        Notification.place(x=450, y=400)

window.grid_rowconfigure(0, weight=1)
window.grid_columnconfigure(0, weight=1)

message = tk.Label(window, text="Face-Recognition-Based-Attendance-Management-System",
bg="SpringGreen3", fg="white", width=50,
                   height=3, font=('times', 30, 'italic bold '))

message.place(x=80, y=20)

Notification = tk.Label(window, text="All things good", bg="Green", fg="white", width=15,
```

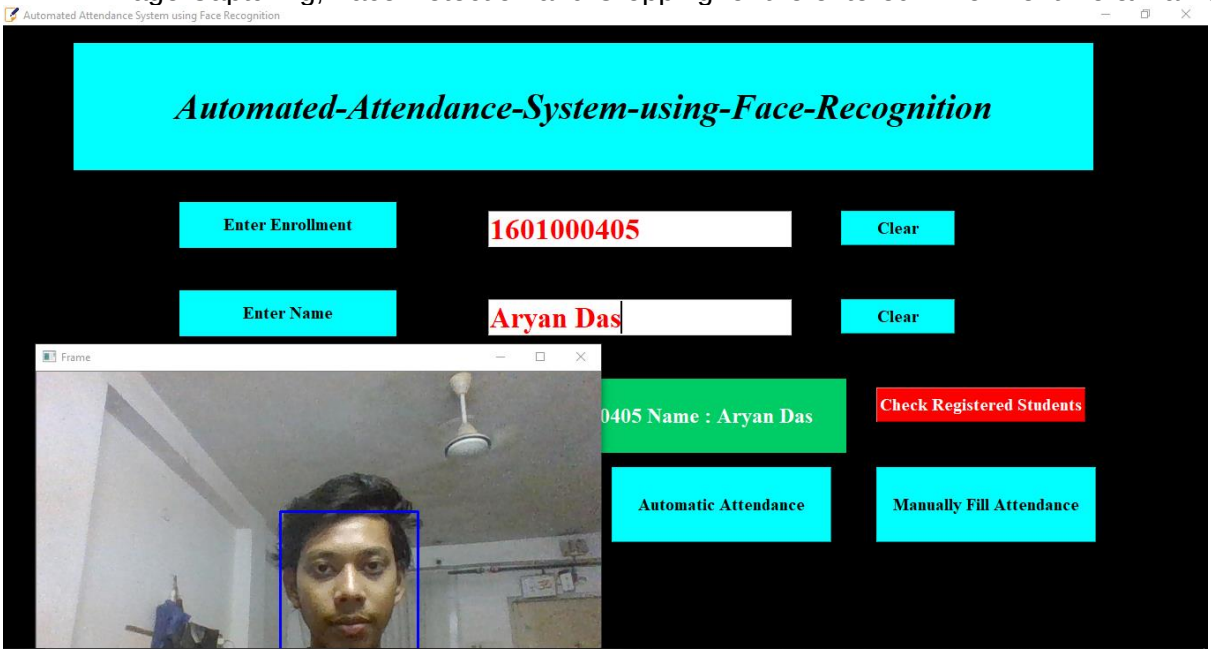
```
            height=3, font=('times', 17, 'bold'))

lbl = tk.Label(window, text="Enter Enrollment", width=20, height=2, fg="red", bg="yellow", font=('times', 15, ' bold '))
lbl.place(x=200, y=200)

txt = tk.Entry(window, width=20, bg="yellow", fg="red", font=('times', 25, ' bold '))
txt.place(x=550, y=210)

lbl2 = tk.Label(window, text="Enter Name", width=20, fg="red", bg="yellow", height=2, font=('times', 15, ' bold '))
lbl2.place(x=200, y=300)

txt2 = tk.Entry(window, width=20, bg="yellow", fg="red", font=('times', 25, ' bold '))
txt2.place(x=550, y=310)

clearButton = tk.Button(window, text="Clear",command=clear,fg="white" ,bg="deep pink" ,width=10 ,height=1 ,activebackground = "Red" ,font=('times', 15, ' bold '))
clearButton.place(x=950, y=210)

clearButton1 = tk.Button(window, text="Clear",command=clear1,fg="white" ,bg="deep pink" ,width=10 ,height=1, activebackground = "Red" ,font=('times', 15, ' bold '))
clearButton1.place(x=950, y=310)

takeImg = tk.Button(window, text="Take Images",command=take_img,fg="white" ,bg="blue2" ,width=20 ,height=3, activebackground = "Red" ,font=('times', 15, ' bold '))
takeImg.place(x=200, y=500)

trainImg = tk.Button(window, text="Train Images" ,fg="white" ,bg="purple2" ,width=20 ,height=3, activebackground = "Red" ,font=('times', 15, ' bold '))
trainImg.place(x=500, y=500)

quitWindow = tk.Button(window, text="Quit", command=window.destroy ,fg="white" ,bg="Red" ,width=20 ,height=3, activebackground = "Red" ,font=('times', 15, ' bold '))
quitWindow.place(x=800, y=500)

window.mainloop()
```

Screenshots:

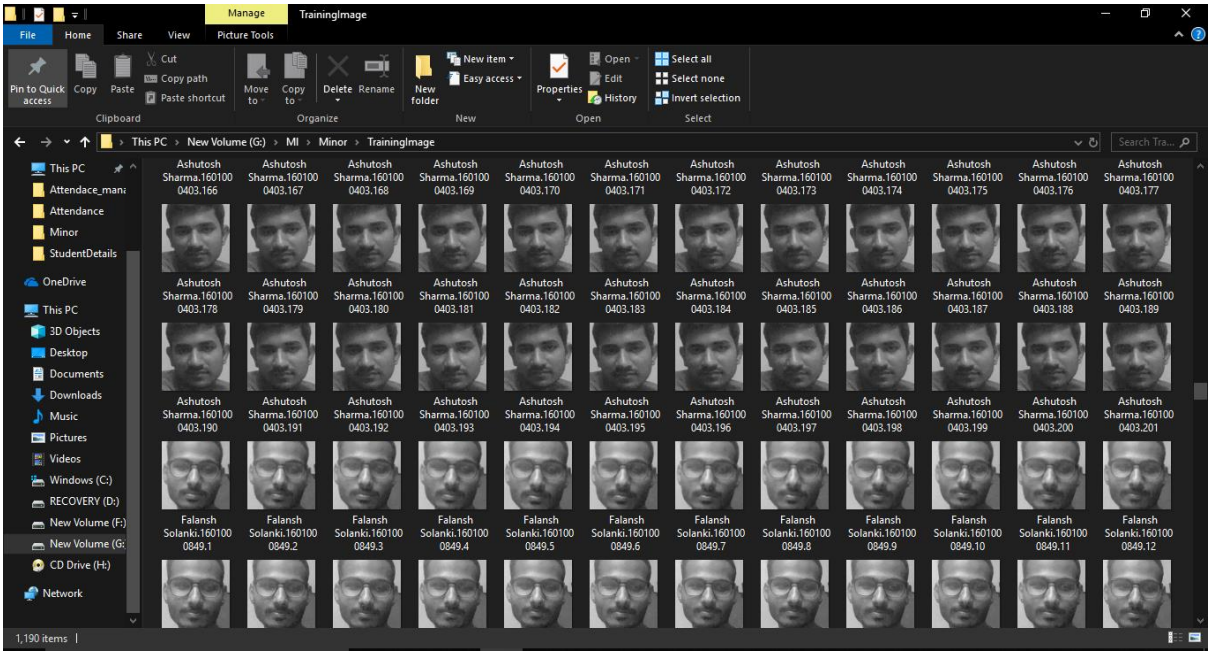
1. Graphical User Interface (GUI)



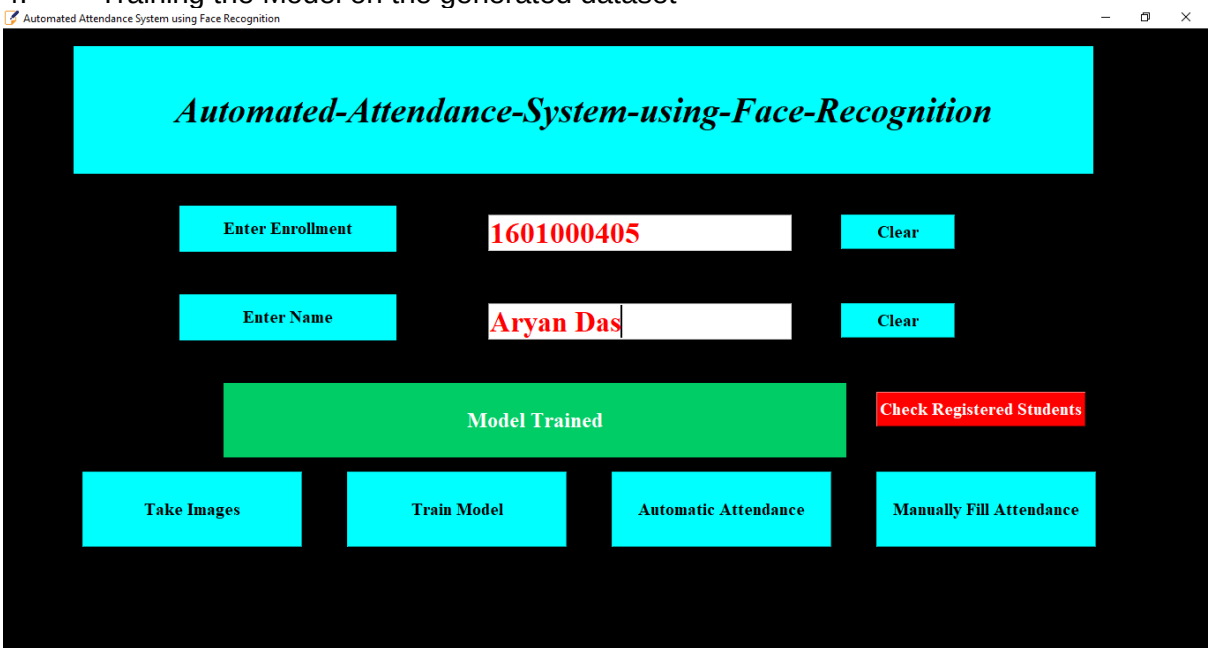
2. Image Capturing, Face Detection and Cropping for the entered Enrollment No & Name



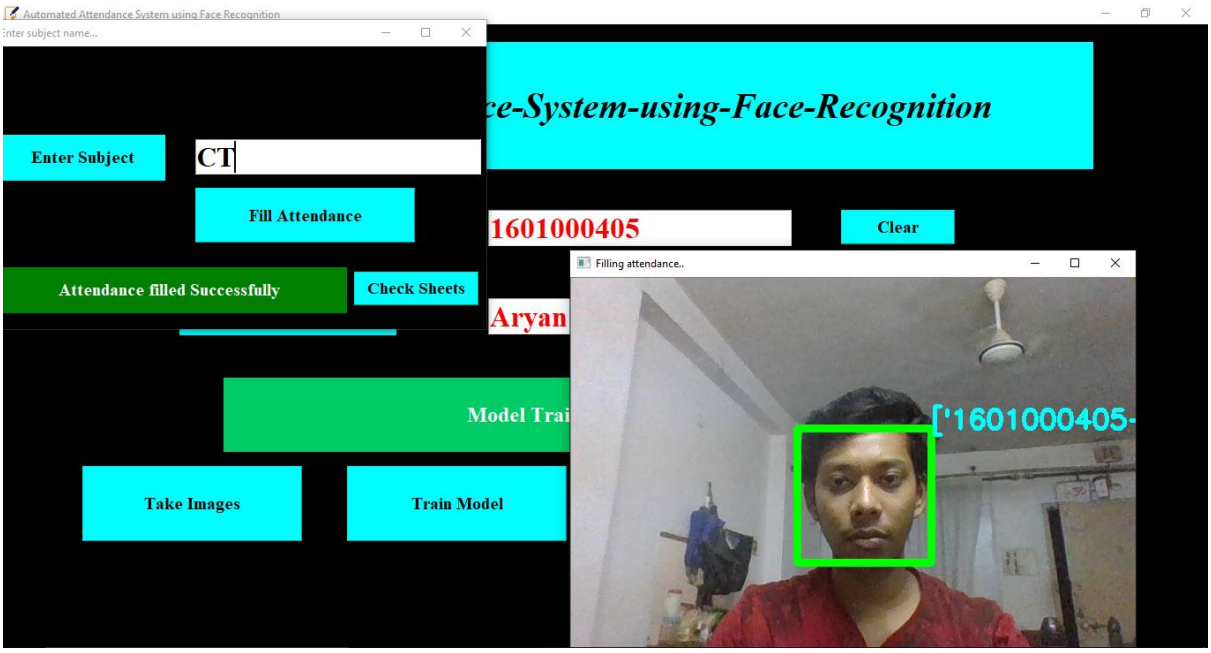
3. Generated Datasets



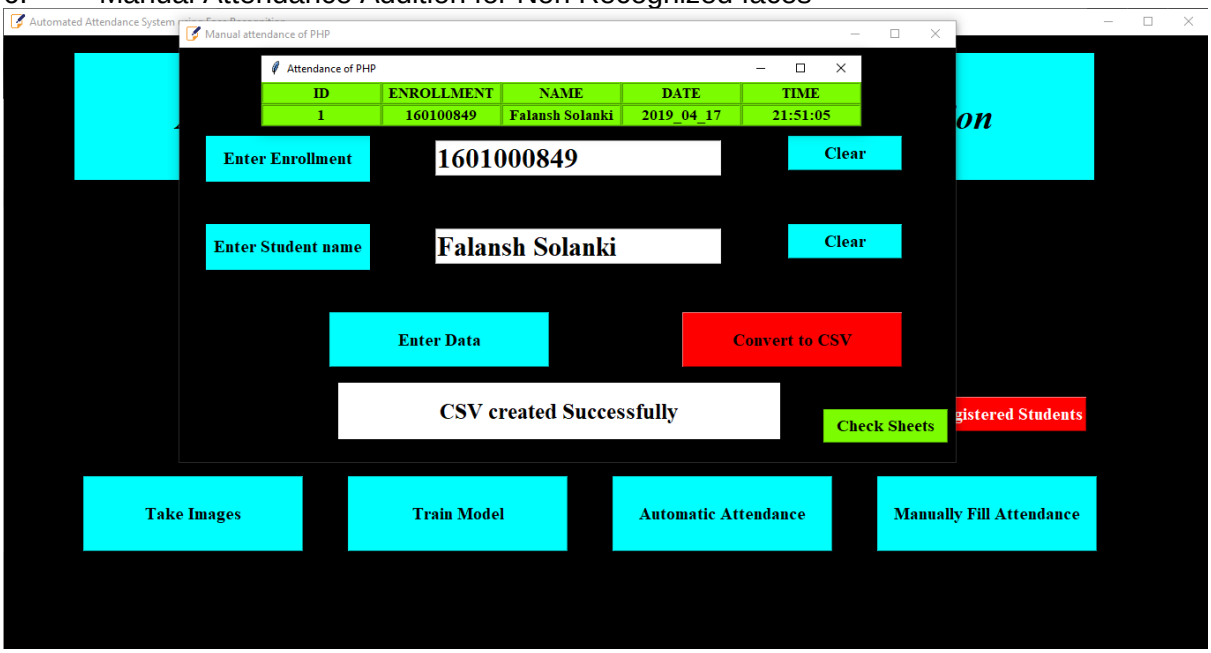
4. Training the Model on the generated dataset



5. Automated Attendance Generation



6. Manual Attendance Addition for Non Recognized faces



6.6 TESTING

Testing is the major quality control that can be used during software development. Its basic function is to detect the errors in the software. During requirement analysis and design, the output is a document that is usually textual and non-executable. After the coding phase, computer program is available that can be executed for testing purposes. This implies that testing not only has to uncover errors introduced during coding, but also errors introduced during previous phases. Thus the goal of the testing is to uncover requirement, design and coding errors in the program.

An elaborate testing of data is prepared and the system is tested using that test date. Errors noted and corrections made during the testing. The corrections are also noted for future use. The users are trained to operate the developed system. Both hardware and software securities are made to run the developed system successfully in future. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately before live operation commences. Testing is vital to the success of any system. System testing makes a logical assumption that if all the parts of the system are correct, the goal will be successfully achieved.

6.6.1 TESTING OBJECTIVES

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an undiscovered error.

A successful test is one that uncovers an as-yet undiscovered error.

Testing Principles

- All tests should be traceable to customer requirements
- Tests should be planned long before testing begins
- Testing should begin “in the small” and progress toward testing “in the large”
- Exhaustive testing is not completely possible
- To be most effective, testing should be conducted by an independent third party

6.6.2 TESTING METHODS

Software Testing Strategies

A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. As important, a software testing strategy provides a road map. Testing is a set of activities that can be planned in advance and conducted systematically.

Various strategies are given below:

- Unit Testing
- Integration Testing
- Validation Testing
- User Acceptance Testing
- System Testing

Unit Testing

Unit testing focuses verification efforts on the smallest unit of software design of module. This is also known as “Module Testing”. Acceptance of package is used for computerization of module. Machine Utilization was prepared and approved by the project leader.

In this testing step, each module is found to be working satisfactory as regards to the expected output from the module. The suggested changes were incorporated into the system. Here each module in the Machine Utilization has been tested.

Integration Testing

After the package is integrated, the user test version of the software was released. This testing consists of testing with live data and various stress tests and result were noted down. Then the corrections were made based on the users feedback. Integration testing is systematic testing for constructing the program structure, while at the same time conducting tests to uncover errors associated within the interface. The objective is to take unit tested modules and build a program structure. All the modules are combined and tested as a whole. Here correction is difficult because the vast expenses of the entire program complicate the isolation of causes. Thus the integration testing step, all the errors uncovered are corrected for the next steps.

Validation Testing

At the culmination of integration testing, software is completely assembled as a package; interfacing errors have been uncovered and corrected, and a final series of software tests - Validation testing - may begin.

User Acceptance Testing

User acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system users at time of development and making changes wherever required.

This is done in regard to the following points:

- Input Screen Design
- On-line Messages to guide the user
- Format of reports and other outputs

After performing all the above tests the system was found to be running successfully according to the user requirements i.e., (constraints).

System Testing

Software is only one element of a larger computer-based system.

Ultimately, software is incorporated with other system elements and a series of system integration and validation tests are conducted. The various types of system testing are:

- Recovery Testing: Many computer-based systems must recover from faults and resume processing within a pre specified time.
- Security Testing: Security testing attempts to verify that protection mechanisms built into a system will in fact protect it from improper penetration.
- Stress Testing: Stress tests are designed to confront programs with abnormal situations.
- Performance Testing: Performance testing is designed to test run-time performance of software within the context of an integrated system.

Black Box Testing

Black box testing is carried out to check the functionality of the various modules. Although they are designed to uncover errors, black-box tests are used to demonstrate that software functions are operational; that input is properly accepted and output is correctly produced; and that the integrity of external information is maintained. A black-box test examines some fundamental aspect of the system with little regard for the internal logical structure of the software.

White Box Testing

White-box testing of software is predicated on close examination of procedural detail providing the test cases that exercise specific sets of conditions and, loops tests logical paths through the software. White-box testing, sometimes called glass-box testing is a test case design method that

uses the control structure of the procedural design to derive test cases. Using white-box testing methods, following test cases can be derived.

- Guarantee that all independent paths within a module have been exercised at least once.
- Exercise all logical decisions on their true and false sides.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structures to assure their validity.
- The errors that can be encountered while conducting white-box testing are Logic errors and incorrect assumptions.
- Typographical errors