CodeVoice

1.1 INTRODUCTION:

Programming environments can create frustrating barriers for the growing numbers of software developers that suffer from repetitive strain injuries (RSI) and related disabilities that make typing difficult or impossible. Not only is the software development process comprised of fairly text-intensive activities like program composition, editing and navigation, but the tools used for programming are also operated textually. This results in a work environment for Programmers in which long hours of RSI-exacerbating typing are unavoidable. Grappling with the syntax of a programming language can be frustrating for programmers because it distracts from the abstract task of creating a correct program. Visually impaired programmers have a difficult time with syntax because managing syntactic details and detecting syntactic errors are inherently visual tasks. As a result, a visually impaired programmer can spend a long time chasing down syntactic errors that a sighted programmer could have found instantly. Programmers facing repetitive stress injuries can have a difficult time entering and editing syntactically detailed programs from the keyboard. Novice programmers often struggle because they are forced to learn syntactic and general programming skills simultaneously. Even experienced programmers may be hampered by the need to learn the syntax of a new programming language.

The primary tool used for programming is a specialized text editor . Early text editors were manipulated entirely via keyboard, and leveraged many keyboard shortcuts for common operations in order to speed the programming process. Unfortunately, keyboard shortcuts are usually composed of an alphanumeric key combined with one or more modifier keys (e.g. Control, Option, Alt), which contributes to RSI when keyed ergonomically. These days, text editors are usually embedded in integrated development environments (IDEs) that provide the programmer with integrated services, such as compilation and debugging .

One way to reduce the amount of typing while programming is to use speech recognition. Speech interfaces may help to reduce the troubles of RSI among computer programmers.

At the other side speech programming may increase access for those already suffering motor impairments. Many disabled programmers are already bootstrapping voice recognition into existing programming environments . However, speech does not map well onto the available applications and programming tasks. Our project uses a principled approach from field of programming languages to allow developers to use speech with much more powerful control.

1.2 PROBLEM STATEMENT:

Software development using programming languages requires keyboard for input and all programming languages are mostly text oriented. The text oriented nature of programming languages is a barrier to persons suffering from arms disability. A person having brilliant mind and potential for programming skills, but suffering from arm injuries or being disabled could not become a programmer. To be a good developer a person must memorize the syntax and keywords of a programming language and should have knowledge of concepts in Computer Science.

1.3 NEED FOR PROPER SYSTEM:

- To enable programmers suffering from arm injuries and visually impaired write code.
- To ease the problem of learning syntax of new programming language.
- Avoid repetitive strain injuries (RSI).
- Reduced time for software development.
- Freeing user form the problem of remembering syntax.
- Enable programmers to use voice recognition.

1.4 OBJECTIVE:

- In our project, we come up with a solution that addresses the above mentioned problems. We propose a methodology for JAVA programming language where a programmer will speak, and code in JAVA will be written accordingly. Structure of special program constructs will also be created simultaneously.

- Our solution requires a microphone attached to the computer. User should have all the basic understanding of JAVA language. He will speak his code that will be in English language but conforms to JAVA syntax and semantics.

1.5 MODULES OF THE SYSTEM:

A handful of packages for speech recognition exist on PyPI. A few of them include:

- apiai
- assemblyai
- google-cloud-speech
- pocketsphinx
- SpeechRecognition
- watson-developer-cloud
- wit

Some of these packages—such as wit and apiai—offer built-in features, like natural language processing for identifying a speaker's intent, which go beyond basic speech recognition. Others, like google-cloud-speech, focus solely on speech-to-text conversion.

There is one package that stands out in terms of ease-of-use: SpeechRecognition.

Recognizing speech requires audio input, and SpeechRecognition makes retrieving this input really easy. Instead of having to build scripts for accessing microphones and processing audio files from scratch, SpeechRecognition will have you up and running in just a few minutes.

The SpeechRecognition library acts as a wrapper for several popular speech APIs and is thus extremely flexible. One of these—the Google Web Speech API—supports a default API key that is hard-coded into the SpeechRecognition library. That means you can get off your feet without having to sign up for a service.

1.6 SCOPE:

- The scope of this project revolves around the development of a code generating system that employs speech recognition , which will allow programmers to write code regardless of their suffering from arm injuries or being disabled.
- It is also a solution for making the process of software development as fast as it will reduce the amount of typing while programming. Also, it will help experienced programmers as there would be no need to learn the syntax of a new programming language.

2.1 EXISTING SYSTEM:

	TOOLS				
FEATURES	**LEGETT AND WILLIAMS**	**PETRY AND PIERCE**	**EMACS LISTEN**	**CACHE PAD**	**CodeVoice**

Punctuation	YES	NO	NO	NO	YES
Existing Symbols	NO	NO	YES	YES	\ YES
New Symbols	NO	NO	NO	NO	YES
Mouse Free Operation	YES	YES	YES	YES	YES
Cross Sr	NO	NO	YES	YES	YES
Cross Editor	NO	NO	NO	NO	YES

Figure 2.1 Literature Survey

2.2 PROPOSED SYSTEM:

Our system is divided into the following modules as shown.

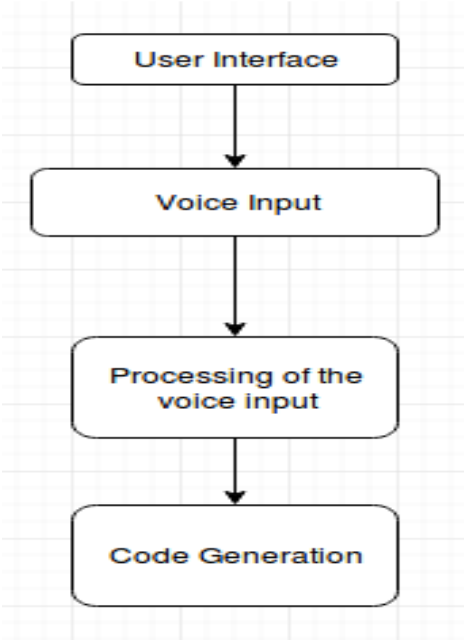

Figure 2.2 System Architecture

- The User Interface module provides an interface for users to interact with the system. Voice to text converter is responsible for converting each listened word to text. A user will speak code using a microphone and a text converter will convert this spoken code to text. For this converted text JAVA syntax and semantics are applied to this text to generate standardized JAVA code.
- The core part of our system is voice to text conversion. Recognizing speech requires audio input, and SpeechRecognition makes retrieving this input really easy. Instead of having to build scripts for accessing microphones and processing audio files from scratch, SpeechRecognition will have you up and running in just a few minutes.
- The SpeechRecognition library acts as a wrapper for several popular speech APIs and is thus extremely flexible. One of these—the Google Web Speech API—supports a default API key that is hard-coded into the SpeechRecognition library. That means you can get off your feet without having to sign up for a service.

- The flexibility and ease-of-use of the SpeechRecognition package make it an excellent choice for any Python project. However, support for every feature of each API it wraps is not guaranteed. You will need to spend some time researching the available options to find out if SpeechRecognition will work in your particular case.
- In this type of speech recognition machine listens to what we say and attempts to translate it into text. The accuracy of dictation ties directly to the CPU's speed and the system's available memory. The more resources, the more contexts that can be considered in a reasonable amount of time, the more likely the resulting recognition will be accurate.All text recognized is then converted to lowercase for the purpose of code generation because JAVA is case sensitive and all the reserved words are in lower case.
- Code generator is the module that actually generates JAVA code from listened words. As a first step, we find a list of the reserved words of JAVA. Now for each reserved word we find words with similar sounds. For example “while” have the following words with similar sounds. "lloyd", "while", "white", "wine" and "voice ".
- After this, for each reserved word we developed a separate list structure of words with similar sounds in Java. When a user speaks a reserved word that word will be converted to text and this text will be matched to the elements of list structure. If a match is found converted text is replaced with that reserved word. If no match is found text is written as it is. Now if this text is wrong and user wants to remove that word user will speak “incorrect”. A list structure is also maintained for same utterances of “incorrect”. If spoken word is matched with that same utterance then that word is removed.At the same time if a match is found and that reserved word has special program construct then that program construct is also generated simultaneously.

2.3 FEASIBILITY STUDY:

The feasibility study is a major factor which contributes to the analysis and development of the system. The decision of the system analyst whether to design a particular system or not depends on its feasibility study. Study of requirement analysis is done through different feasibility study. Feasibility study is undertaken whenever a possibility of probability of improving the existing system or designing a new system. Feasibility study helps to meet user requirements.

It enables us to determine the potential of existing system and improving it. It helps to develop a technically and economically feasible system. It helps to know what should be embedded in the system. It also helps to develop a cost-effective system. The project concept is feasible because of the following:

2.3.1 Technical Feasibility

2.3.2 Economical Feasibility

2.3.3 Operational Feasibility

2.3.1 TECHNICAL FEASIBILITY:

- The installation of the software requires setup of Desktops and a connected microphone.
- Each System needs to have installed Java Development Kit and Python which are both open source.

2.3.2 ECONOMICAL FEASIBILITY:

Economical feasibility means whether a business or a project is feasible cost wise and logistically. Economists calculate economic feasibility by analysing the costs and revenues a business would incur by undertaking a certain project.The implementation of the solution requires no additional costs to the existing text oriented systems.

2.3.3 OPERATIONAL FEASIBILITY:

- It is a measure of how well the system will work in the organization. It is also a measure of how people feel about the system/project. In this project the user feels that the system is very user friendly.
- The desktop application is easy to use and it is expected that the user/programmer knows basic syntax of Java Programming Language and concepts of Computer Science.

3.1 METHOD USED FOR REQUIREMENT ANALYSIS:

INCREMENTAL MODEL

Incremental development is based on the idea of developing an initial implementation, exposing this to user comment and evolving it through several versions until an adequate system has been developed. Specification, development, and validation activities are interleaved rather than separate, with rapid feedback across activities.

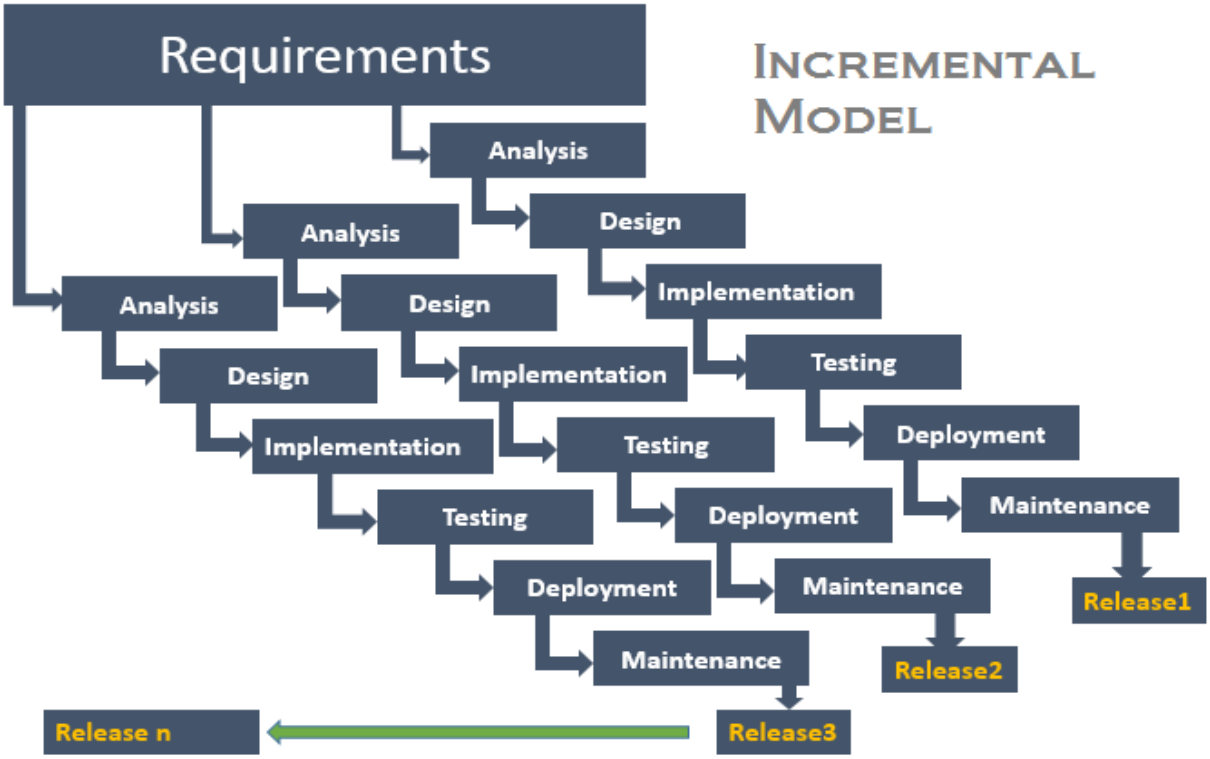

Fig 3.1 Incremental Model

3.2 DATA REQUIREMENTS:

It is a measure of the how practical solutions are and whether the technology is already available within the organization. If the technology is not available to the firm, technical feasibility also looks at whether it can be acquired.

Technical feasibility centers around the existing system and to what extent its support can be extended to the proposed system. This project is technically feasible and to maximum extent the existing systems support the proposed system.

3.3 FUNCTIONAL REQUIREMENTS:

It is a measure of the cost-effective of a project or solutions. It is a measure of whether a solution will pay for itself or how profitable a solution will be, this is often called a cost-benefit analysis.

3.4 NON FUNCTIONAL REQUIREMENTS:

It is a measure of the cost-effective of a project or solutions. It is a measure of whether a solution will pay for itself or how profitable a solution will be, this is often called a cost-benefit analysis.

3.5 SYSTEM SPECIFICATIONS:

This specification describes the internal interfaces between different parts of computer platform firmware. This allows for more interoperability between firmware components from different sources.

3.5.1 Hardware specification:

The Hardware tools used in the project are:

1. RAM: 128 MB; 64 MB for Windows XP (32-bit)
2. Disk space: 124 MB.
3. A microphone or an earphone

3.5.2 Software Specification:

The Software tools and IDEs used in the project are :

1. JDK 8.1
2. Python IDLE 3.7

4.1 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 the system properties. Conceptually every SRS should have the components:

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

4.1.1 GLOSSARY:

- Java is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible. It is intended to let application developers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but it has fewer low-level facilities than either of them.

- Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

4.1.2 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. Use cases are not only texts, but also diagrams, if needed. The purpose of use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Because other four diagrams (activity, sequence, collaboration and State chart) are also having the same purpose. So we will look into some specific purpose which will distinguish it from other four diagrams. 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.

So in brief, the purposes of use case diagrams can be as follows:

- Used to gather requirements of a system.
- Used to get an outside view of a system.
- Identify external and internal factors influencing the system.
- Show the interacting among the requirements are actors.

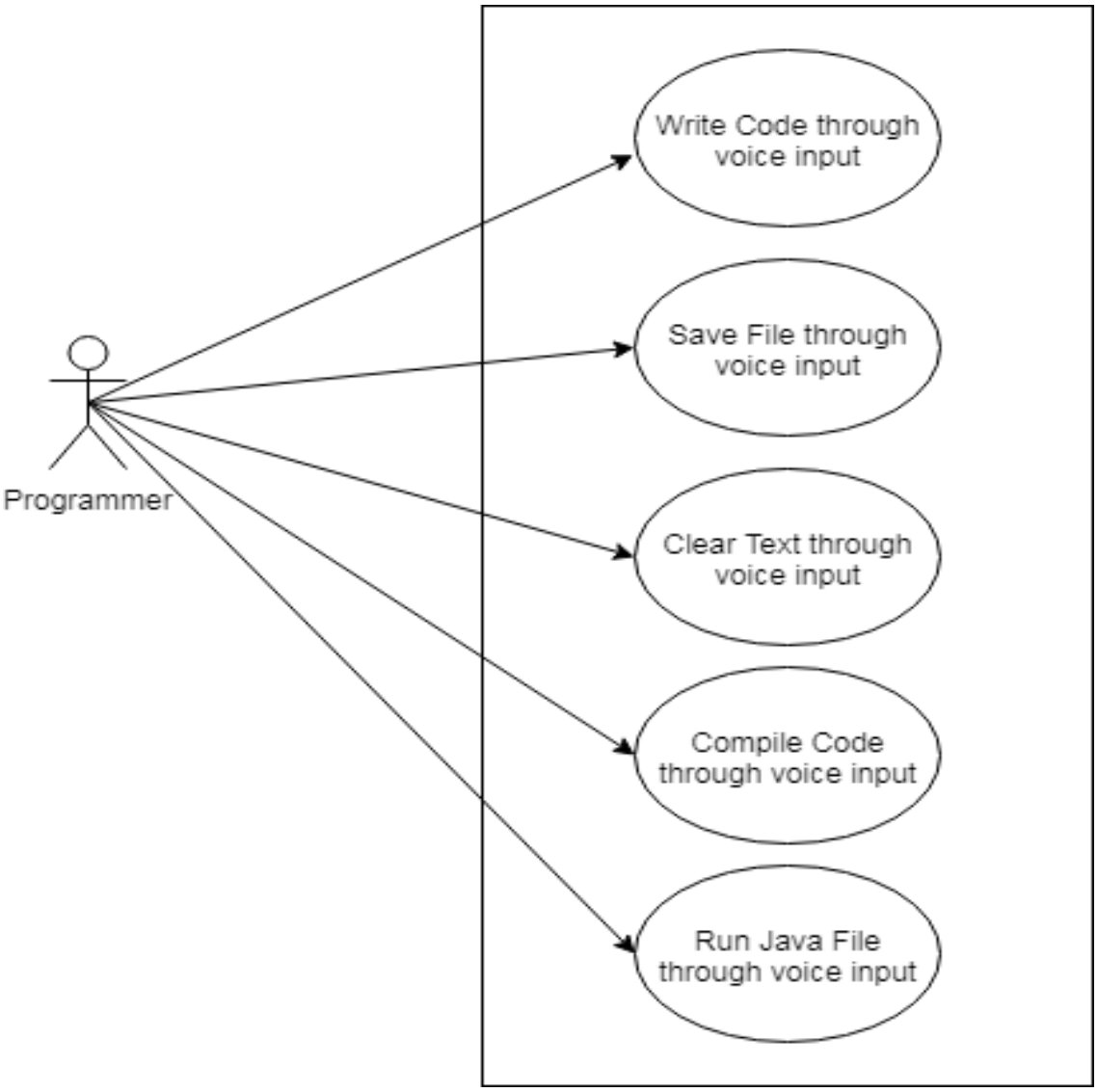

Fig 4.1 Use case diagram

4.2 Conceptual Level class diagram:

In software engineering, a class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. The class diagram is the main building block of object-oriented modelling. It is used both for general conceptual modelling of the systematics of the application, and for

detailed modelling translating the models into programming code. Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

In the diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class. It is printed in bold and centred, and the first letter is capitalized.
- The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
- The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

In the design of a system, a number of classes are identified and grouped together in a class diagram that helps to determine the static relations between them. With detailed modelling, the classes of the conceptual design are often split into a number of subclasses.

4.3 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 a system behavior. An activity diagram provides a complete specification of a behavior and not, like the interaction diagrams, a single possible scenario.

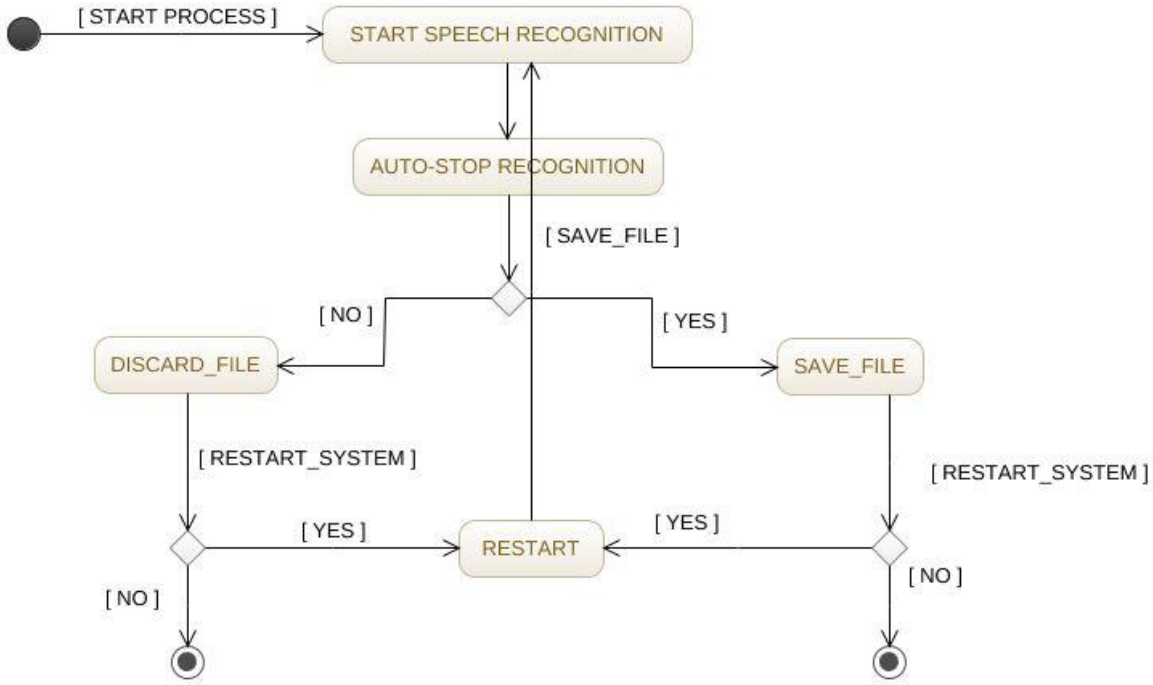

Fig 4.3 Activity Diagram

4.4 Data flow Diagram(Level 0,1)

Planning Managerial Issues is the process of identifying and resolving issues. Problems with staff or suppliers, technical failures, material shortages – these might all have a negative impact on your project. If the issue goes unresolved, you risk creating unnecessary conflicts, delays, or even failure to produce your deliverable.

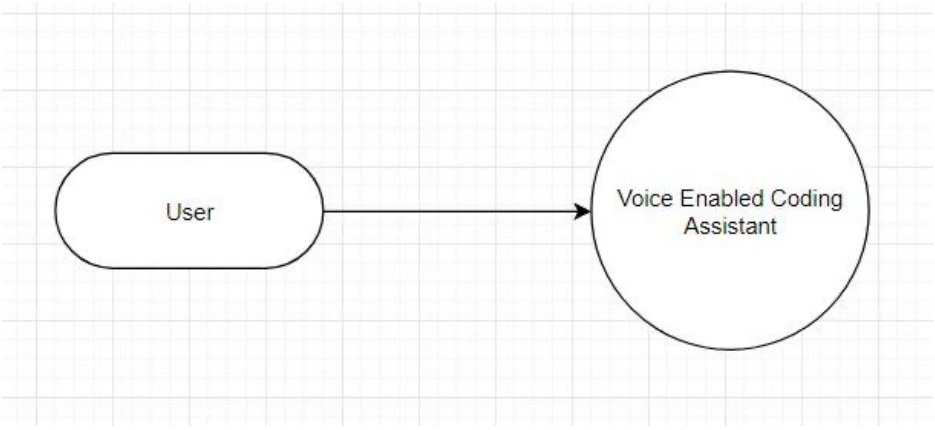

Fig 4.4.1 DFD Level 0 Diagram

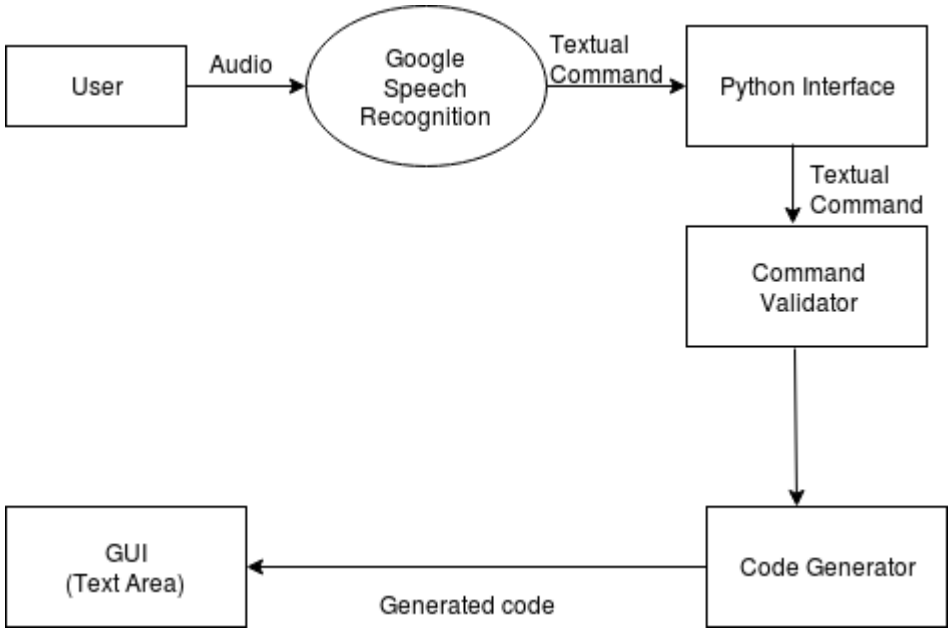

Fig 4.4.2 DFD Level 1 Diagram

5.1. DETAILED CLASS DIAGRAM:

The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object oriented systems because they are the only UML diagrams which can be mapped directly with object oriented languages. The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a structural diagram. The purpose of the class diagram is to model the static view of an application.

The following points should be remembered while drawing a class diagram:

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified.
- For each class minimum number of properties should be specified. Because unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. Because at the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

5.2. INTERACTION DIAGRAM:

From the term Interaction, it is clear that the diagram is used to describe some type of interactions among the different elements in the model. This interaction is a part of dynamic behavior of the system.

This interactive behavior is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram. The basic purpose of both the diagrams are similar.

Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

The purpose of interaction diagram is –

- To capture the dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.

5.2.1. SEQUENCE DIAGRAM:

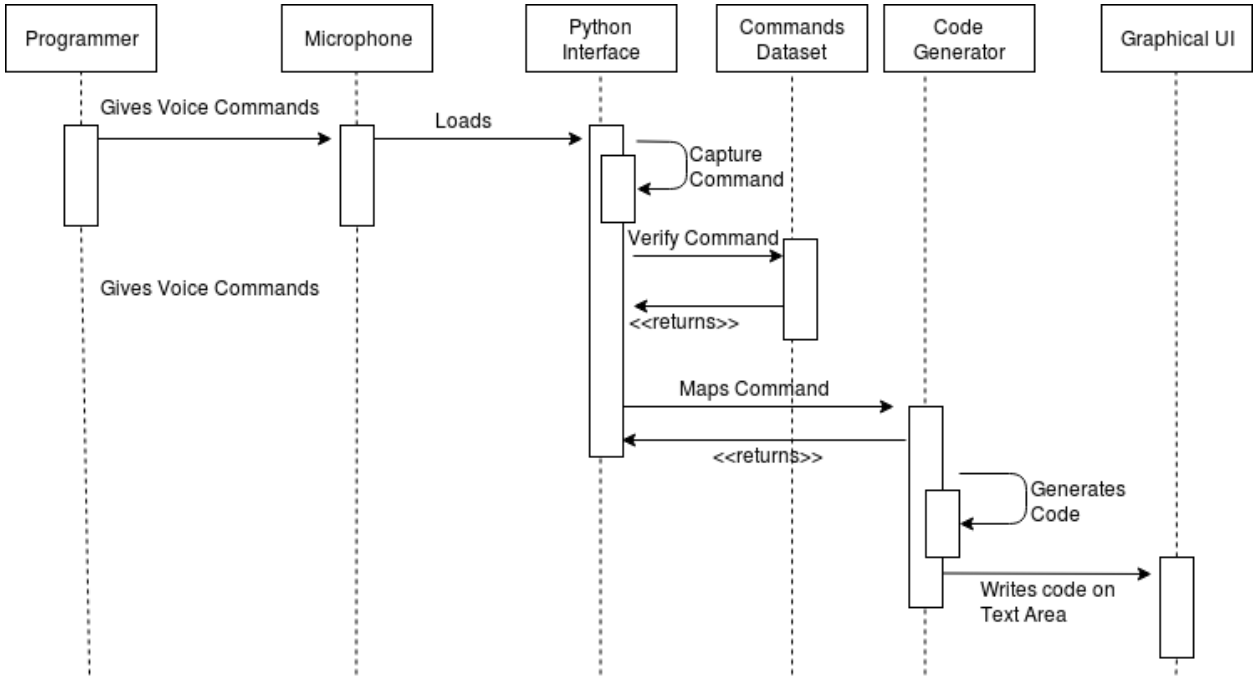

Figure 5.2 Sequence Diagram

5.3 STATE DIAGRAM:

A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioral diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as State machines and State-chart Diagrams. These terms are often used interchangeably. So simply, a state diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli.

We can say that each and every class has a state but we don't model every class using State diagrams. We prefer to model the states with three or more states.

5.4. ACTIVITY DIAGRAM:

Activity diagrams illustrate the dynamic nature of a system by modelling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation.

Activity diagrams are used to model workflow or business processes and internal operation. The figure shows the work flow the system. The client logs in, gives the requirements, Admin views the requirements, contacts the Dealers and then finally generates the final product.

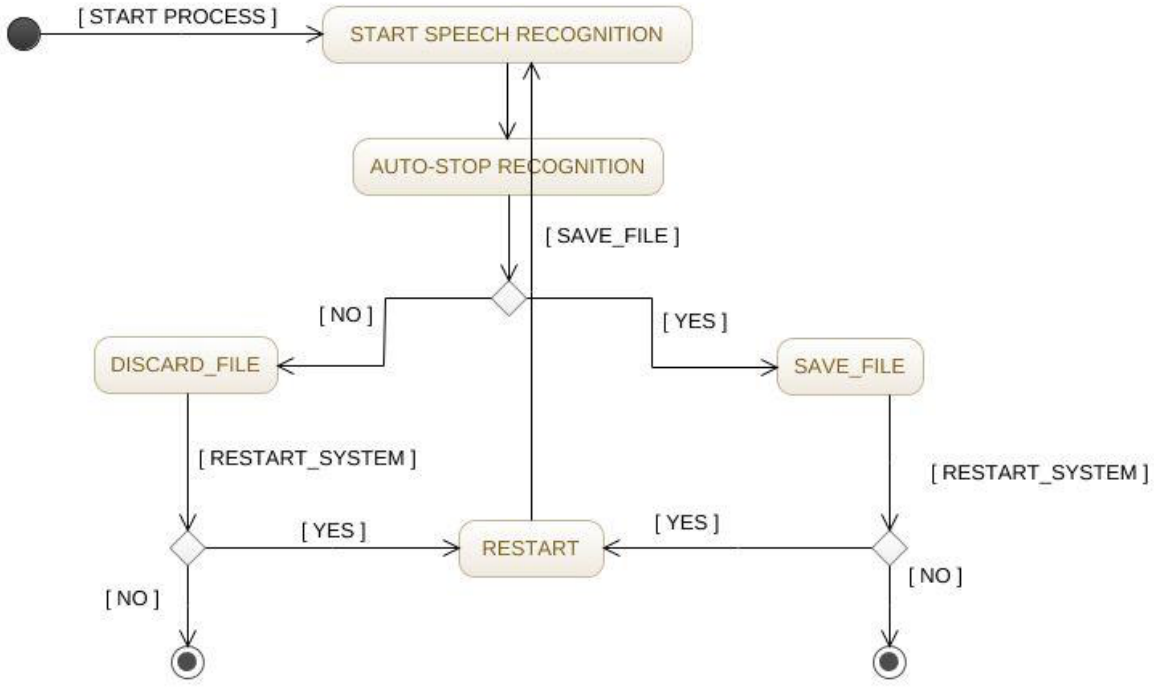

Figure 5.6 Activity Diagram

5.5. OBJECT DIAGRAM:

- Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.

- Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams. Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.
- Object diagrams are used to render a set of objects and their relationships as an instance.

The purpose of the object diagram can be summarized as –

- Forward and reverse engineering.
- Object relationships of a system
- Static view of an interaction.
- Understand object behaviour and their relationship from practical perspective

5.6. COMPONENT DIAGRAM:

- Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now the question is, what are these physical aspects? Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.
- Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.

The purpose of the component diagram can be summarized as –

- Visualize the components of a system.
- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

5.7 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.

5.7.1 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

5.7.2 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.

Speech to Text unit testing checks how accurately speech models transcribe user utterances into text. This is accomplished with a set of ground truth that includes snippets of audio and their corresponding transcriptions in text form. The model is trained and evaluated on subsets of the ground truth and then we evaluate the results looking for specific patterns.

The test results include a list of each input and transcription output as well as a description of any errors and a word error rate (WER) summary statistics.

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

6.1 LIMITATIONS OF PROJECT:

- Requires internet connectivity.
- Hardware limitations: The microphone and the PC must be present physically within the range limits of the network.
- Interface to other applications: This application cannot be used inside or in association with any other application.
- Parallel operation: The application cannot connect with more than one microphone parallel.

6.2 FUTURE ENHANCEMENT:

- This approach can be extended to :all textual programming languages and also to visual programming languages.
- This approach can also help in query writing.
- It will help experienced programmers as there would be no need to learn the syntax of a new programming language.

7.1 REFERENCE BOOKS:

[1] Joseph and Harvey, Brandon. NaturalJava: A Natural Language Interface for Programming in Java

[2] Damper, B. and Leedham, G. Human factors. In Speech Processing. McGraw Hill, NRC, Ottawa.

7.2 OTHER DOCUMENTATIONS AND RESOURCES:

[3] https://www.academia.edu/7440380/Hands_free_JAVA_Through_SpeechRecognition

[4] Desilets, Alain. VoiceGrip: A Tool for Programming by Voice.

[5] https://www.py4j.org/getting_started.html