Ok Eclipse+: Enabling Voice Input to augment User Experience in Integrated Development Environment

Srujan Barai
NC State University
sjbarai@ncsu.edu

Daxkumar Amin
NC State University
dkamin@ncsu.edu

Aayushi Agrawal
NC State University
agrawa@ncsu.edu

Shenee Ashara
NC State University
spashara@ncsu.edu

ABSTRACT

In this world of smart devices, some IDEs are still old-fashioned. Developers have always been looking for making editors smarter to get the most out of them. OkEclipse is one those projects that takes IDE experience to an entirely new level. Ever thought about talking to your IDE? Wouldn't it be great to have certain tasks, within your IDEs, be automated using just your voice commands? That is exactly what OkEclipse does. But well, there are few flows with this plugins, for example: (i) It is slower compared to keyboard shortcuts. (ii) The speech recognition accuracy is very questionable. (iii) Users have to remember all the voice commands, just like keyboard shortcuts. Keeping all these in mind, we decided to make OkEclipse+ which would solve all the issues mentioned above. The main features about OkEclipse+ is it's Fix command. The main idea behind the fix command is to help developers, get rid of trivial issues with just one command, in a blink of the eye. The Fix command, as you might expect, works on the editor level errors and not the logical errors. We will talk ahead in details about how it works, it's amazing scope and some of its limitations.

Keywords

Voice recognition, Integrated development environments, Q&A repositories, Eclipse plugins, IDE Errors

1 INTRODUCTION

In the era of abundant voice libraries, it was felt that IDEs are not taking advantage of this facility. While using an IDE, users rely on mouse, keyboard and touch displays as means of giving input to the system. Thus, it was realised that voice can be considered as a potential source input when it comes to IDEs. This would reduce the time consumption as well as reduce the work to be performed by the user. But, according to an evaluation conducted by expert Java developer it was concluded that having only voice as a source of input would in fact decrease the time efficiency. [10] Thus, using voice commands for simple tasks for example accessing commands from menu bar would prove more efficient. Taking the above facts in consideration "ok Eclipse" was made as a plugin for Eclipse IDE which listens to user and accesses the menu for them. Thus, user won't be required to remember the shortcuts or where the command is present on menu bar, ultimately increasing the time efficiency. Also a Q&A section has been implemented which takes message from console logs and queries stackoverflow repositories for appropriate solutions.

On an initial survey of 30 developers who uses eclipse in their daily life, it was found out that developers would prefer remembering shortcuts over speaking them to do things as it would result in same time consumption because instead of remembering the shortcut they will have to remember the speech commands.Rather than being discouraged by this fact, we took it as an inspiration and concluded that users want a speech command that would increase their time efficiency. To put this conclusion in action we decided to take OkEclipse to another level where it can do things beyond refactoring and declaring a new variable. The most annoying thing for a developer is fixing errors. Some of these are logical and hence depends on the logic of the code but rest of the errors can be solved without depending on the code. To solve these kind of errors we are planning to introduce a magical keyword "Fix". This keyword would work in any context, meaning irrespective of what context the error is, the error can be fixed with this magical keyword.

To summarize, the keyword - "FIX" plans to solve issues like: if user is developing Android applications using Eclipse (although which now is deprecated), there are some APIs that are restricted to only new versions of Android SDK. In this case, there will be some API calls that does not work with older Android versions. In such scenarios, the "FIX" command will modularize code according to the SDK levels to make sure certain code only runs on SDKs that facilitates those APIs.

2 PREVIOUS WORK DETAILS

The first group detected the problem that user has to do a lot of work while working in IDE. By adding voice commands and automatting some functions in IDEs greater efficiency can be achieved. Now the reason to choose Eclipse as an IDE to implement project is because its open source and is formed of various plugins and thus, creating a plugin for speech detection for current version of Eclipse make sense. They addressed and solved following two problems:

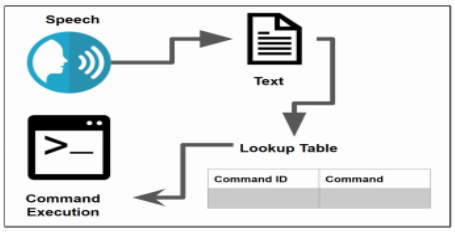


Figure 1: Listening Menus

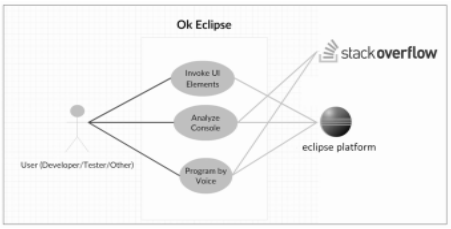


Figure 3: Use Cases

1. **Listening Menus:** Eclipse is composed of various plugins for editor, views and menus which are driven by specific set of commands. Navigating views with a mouse is cumbersome especially in scenarios where certain menus require traversing multiple levels down the menu. This is solved by identifying every menu item associated with a command. Now each command has a unique command ID and thus, creating a look-up table which maps them will lead to execution of that command. Now to convert speech to text CMU Sphinx is used. Here, the user is expected to recite the label text and the engine will invoke the appropriate command using the look-up table. The working of this functionality can be understood from the following figure:

2. **Loud Console:** In Eclipse, the console portrays exact behavior of code at the given time- Errors, user defined messages and exceptions. The first group has recommended fixes by extracting the relevant text of interest from console log to form a query and mine Q&A (stackoverflow.com) repositories for appropriate solutions. Whenever the user feels like he might need a recommendation, he/she can invoke the 'Find' command to get a suitable recommendation for the relevant console log displayed at a given time. The working of this functionality can be understood from Figure 2 and to incorporate all these features they have considered following use cases which can be followed from Figure 3:

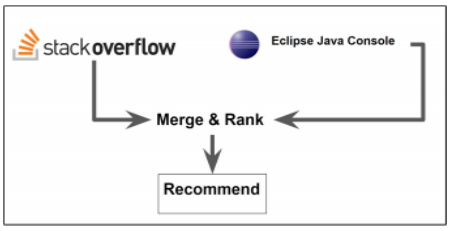


Figure 2: Loud Console

3 LITERATURE REVIEW

Efforts to implement speech recognition for programming tasks using conventional natural language processing tools have had limited success. Jeff Gray speech enabled the Eclipse programming environment [9], but not the editor. Speech enabled IDEs is almost one step to making a usable programming environment. To edit and navigate through code by voice, developers need to speak code snippets of the program text mixed with navigation, editing, and transformation commands. VoiceCode[10], uses finite state command grammars to provide support for Python and C++. But VoiceCode has not yet been formally evaluated. Speech recognition systems allow human machine communication to acquire an intuitive nature that approaches the simplicity of interhuman communication.

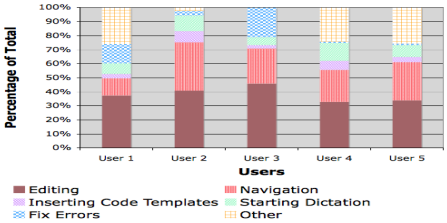


Figure 4: Distribution of Spoken Java commands

Small vocabulary speech recognition dictionary would allows only a small number of words need to be recognized. The recognition of words from the small dictionary can significantly benefit by the use of robust voice operated control components, as they would facilitate the interaction with their users and render it much more reliable due to less number of words to be recognized. Figure from Begel, A., & Graham [8] gives a breakdown of the distribution of Java commands used for different purposes. The commands majorly spoken were for editing and code template insertion. Editing commands were used in place of string editing features that include deleting a character from a word, inserting spaces between words, changing the capitalization of a word. Research says that 50 percent to 75 percent of the cost of software development is spent on debugging [11]. While many tools for finding bugs and reporting error traces exist, but the issue of automatically localizing and resolving the bug is far less understood, and constitutes an active research field.

4 PREVIOUS SURVEY

1. Which one would you prefer - A complex keyboard shortcut or voice command for a task?
We asked this question to gauge how many developers would prefer voice commands for IDE. This gave a rough estimation of how many many people would be interested in us-

ing the Ok Eclipse plugin. The survey results show that 50 percent of the survey takers prefer voice commands over complex keyboard shortcuts. In a powerful IDE such as Eclipse there are many functionalities and respective keyboard shortcuts. Remembering a large number of keyboard shortcut can be a cumbersome task. And hence simple voice commands can be very helpful.

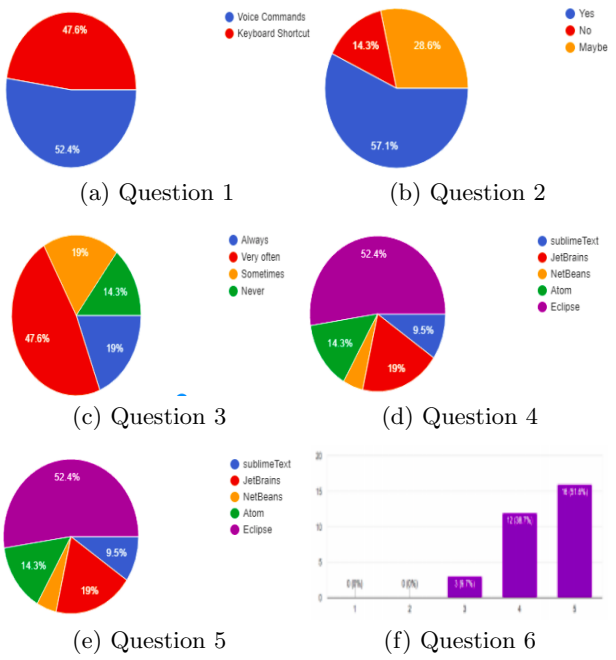
2. Would you like Ok Eclipse to recognise a larger set of commands with a little trade off to response time?
This questions was asked to know if users would be interested in additional commands to the already existing version of Ok Eclipse.
There can be a trade-off between the number of commands supported by the system and the response time. Larger set of commands can increase the response time, as it would take more time to map the voice signal to corresponding text in the Spinx library. However, our survey responses indicate that 85 percent of the users want to have more voice commands in the plugin, which would make various tasks for them easy.

3. In comparison to other IDEs how much do you prefer using Eclipse?
This question would make us understand that how often developers use Eclipse. This would also help us understand the usefulness of the application. The survey results show that 70 percent developers use Eclipse very often. Also, Eclipse was ranked most popular IDE in 2017. Total market share of Eclipse is around 23 percent. Hence, Ok Eclipse would be quite helpful to the developers in general.

4. Which IDE do you use the most?
We were interested to know what other IDEs are popular among the developers. This would help us to analyze to what other IDEs we can extent this application.The results of the survey show that Eclipse is by far the most popular IDE. We only included the IDEs that are open source. JetBrains and Atom are the next best IDEs.

5. Would you prefer a two way communication with the IDE?
This question was asked to evaluate the need of one of the functionality, we are going to implement. Two way communication here means that when you provide a voice command to the application, it give an appropriate voice response. Around 55 percent of the survey takers say that they want the plugin with two-way communication.

6. How cool would it be - if you can solve the errors in your code just by saying 'Solve error'
This question was again to evaluate the popularity of one of the features we are going to implement. Eclipse does provide the suggestions, if there are any syntactic of structural error in the code. We are trying to leverage that functionality and provide a voice command over it.More than 85 percent of the users rated this feature 4 and above - on scale of 1-5.



(a) Question 1 (b) Question 2

(c) Question 3 (d) Question 4

(e) Question 5 (f) Question 6

Figure 5: Previous Survey

5 EXPERIMENTS

etc. Try and incorporate as many figures as you can for this.) To enhance the previously implemented project, we had few ideas in our mind. But we had to evaluate them considering the time constraint for the project. We enumerated following ideas and measured pros and cons for them to decide which one would be the coolest to implement in a given time span:

1. **Two way communication with IDE:** Our first thought was to implement a two way communication with IDE where in the IDE would also suggest options in speech commands. This idea is feasible and it can also have applications for individuals with eyesight disability. For example the user gives command to search a particular word and if it's not found then the IDE would speak back to user that "word not found". This idea seems really cool but on going through materials regarding bidirectional communication we found it really difficult to approach implementation. And thus, due to time constraint we couldn't implement this idea.
2. **Use of speech command to draw UML diagrams:** Eclipse provides way to facilitate plugin named 'UML Designer' to draw various UML diagrams in Eclipse IDE. UML is a standard for specifying, constructing and documenting elements of object-oriented programming projects. This plugin allows users to draw various UML diagrams like class diagram, data flow diagram, sequence diagram, etc. It also lets user drill down into tree representation of XML defining the UML. Our idea was to leverage the functions of this plugin and map them with voice command through which the user just needs to keep on describing his/her diagram in words and doesn't have to touch keyboard or mouse. We studied the 'UML Designer' plugin's working and

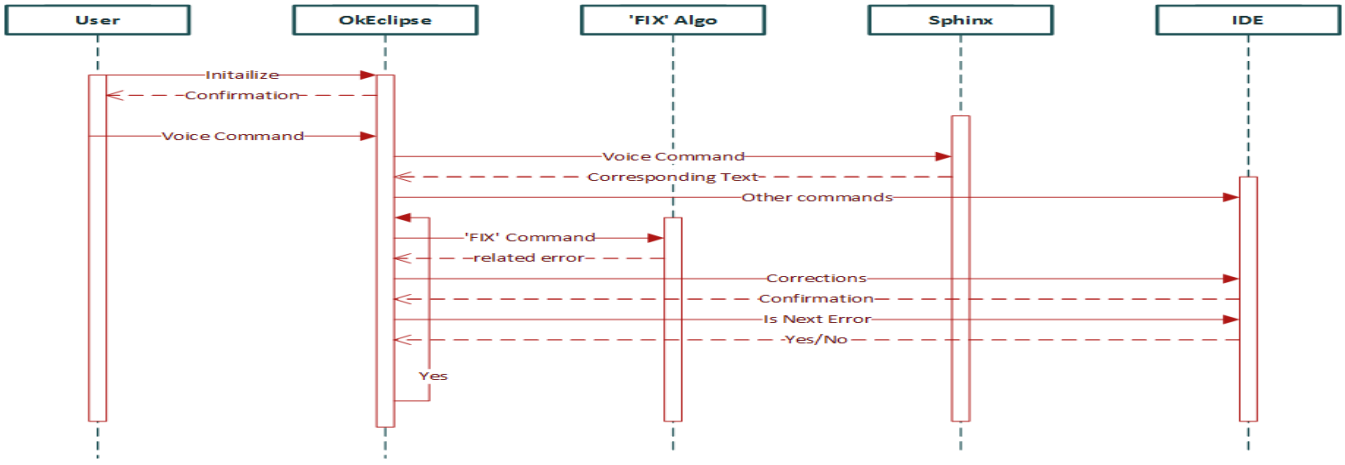


Figure 6: Sequence Diagram

how it maps the functions to execution of drawing diagrams. But again due to time constraint we decided that we won't be able to deliver a complete product for this as well.

3. **Implementing speech recognition for some other IDE other than Eclipse:**

4. **Using NLP:** When we give users a set of command they feel restricted and instead for remembering where the functionality is present on Menu bar they need to remember the speech command for given functionality. Thus, it doesn't fix the problem that we are trying to solve. Natural Language Processing (NLP) and Context recognition can be added and keyword can be extended from user's statement and given to lookup table to map with executable commands. We couldn't find much here because for an IDE the user will use a lot of technical and words specific to certain domain. Here, to solve this problem we came across Natural Language Programming (NLP) which is an ontology-assisted way of programming in terms of natural language i.e. English. A structured document with Content, sections and subsections for explanations of sentences forms a NLP document, which is actually a computer program.

5. **Automatic error resolution using voice command "FIX":** Finally after exploring the above ideas and their feasibility we decided on adding just one extra command "FIX" in the previous implementation which on listening to command "FIX", would internally take suggestions that Eclipse provides when cursor on error and implement them solving the error. For example, if user calls a function which he/she hasn't defined, then on calling "FIX" the plugin would automatically detect this error and define the function for user and the user just needs to later add the logic for this function. The working and technical details of this approach are explained further in the report.

6 IDEA

Our idea is to implement automatic solving of basic errors which uses the first implementation of project as base and implements on top of it.Now to understand the commands, we are taking the speech vector and giving it to CMU sphinx library which converts the vector to text. If this text contains the pre-decided keyword "FIX", it will look for a function that returns the possible solution provided by Eclipse for the current error displayed. If this returns only one solution to the error, the command for that solution is executed. Alternatively we have also considered a scenario where more than one solution is possible in which case, all these solutions are presented to the user by editor and the user will be given choice to choose whatever they finds useful. Once the user selects the solution, command for that is executed.

7 CLASS DIAGRAM

Class diagram is a type of static structure diagram that describes the structure of our system by showing it's classes, attributes, operations (or methods), and the relationships among objects. It is the main building block of object-oriented modelling and used for general conceptual modelling of the application and for translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed. Class diagram of OkEclipse+ is shown in figure[].

8 USE CASE DIAGRAM

Use case diagram is a behavior diagram which describes a set of actions that a system can or should perform when collaborated with users i.e. actors. These set of actions are called use cases. They basically describe the functionality of application that the user can perform. Use cases will provide some result to either the user or to any stakeholder of the application. Thus, use case diagrams are used to associate actors or users to different set of actions that the application can perform.

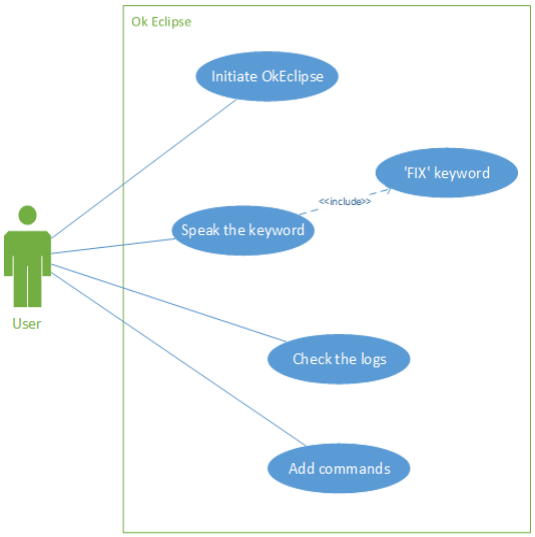


Figure 7: Use Case Diagram

9 SEQUENCE DIAGRAM

A software sequence diagram is used to describe interactions of objects in association with time. It depicts the objects and classes involved in the our application and the sequence of messages exchanged between these objects which are needed to carry out the functionality. The parallel vertical lines represent lifelines of objects and the horizontal arrows represent the messages exchanged in order of their occurence. The sequence diagram of our application is as follows:

10 FEATURES

The best feature of OkEclipse+ is its capacity to solve 224 different types of errors. Here we won't be able to list and explain all various types of errors that it resolves but we have illustrated few of them below. Also if you wish to know all kind of errors that "FIX" command solves, please visit:
Examples of errors that OkEclipse+ resolves are as follows:
`http://tiny.cc/OkEclipseErrorList`

1. case.IProblem.UndefinedMethod: This is a basic mistake sometimes programmers make. It might happen that in hurry the user forgets to write the method or decides to write the method later and call it without defining it. Here Eclipse shows error and this type of error is solved by voice recognition. It defines the method so that the user can move forward and run the program without any error and can later write the logic for the method.

2. case.IProblem.TypeMismatch: When type mismatch error occurs, "FIX" command will automatically correct it by using type casting to appropriate type that is correct for the variable.

3. case.IProblem.UnhandledException: To handle an exception OkEclipse+ adds a try-catch clause and adds the part of program which throws exception inside this try-catch clause.

4. case.IProblem.MissingReturnType: When certain method is written with a return value and its return type isn't mentioned while defining the method, Eclipse shows an error. This error is solved by our project by setting return type of method as the type declared of the variable that it is returning.

Apart from solving so many errors with just one voice command one more interesting feature is Sphinx library that is being used for speech-to-text conversion. Sphinx have many advantages over other speech-to-text libraries. Sphinx is based on state of art speech recognition algorithms for efficient speech recognition. Its tools are designed specifically for low-resource platforms. Sphinx has a flexible design and support several languages like US English, UK English, French, Mandarin, German, Dutch, Russian and ability to build models for others. Due to this the same library can be used if in future multilingual voice recognition support is to be added in Eclipse. Sphinx also has wide range of tools for many speech recognition related purposes like keyword spotting, alignment, pronunciation evaluation etc. Thus, using Sphinx library opens lot of future scope opportunities for enhancing on same platform as Sphinx had large number of applications.

11 TECHNOLOGY USED

For this project everyone of had worked on Eclipse before but none of us were aware of the architecture of Eclipse and the flow of its plugins. Thus, we decided to keep the technology stack pretty simple and basic. We used the same technology stack used by the first group with addition of Eclipse Plugin Development Environment which is basically used "FIX" command. We are leveraging following technologies for OkEclipse+:

1. **CMU Sphinx:** It is a group of speech recognition systems developed at Carnegie Mellon University. These include a series of speech recognizers (Sphinx 2 - 4) and an acoustic model trainer (SphinxTrain). It is a continuous-speech, speaker-independent recognition system making use of hidden Markov acoustic models (HMMs) and an n-gram statistical language model. Sphinx features feasibility of continuous-speech, speaker-independent large-vocabulary recognition. We are using CMU Sphinx to detect the voice command given by user which can be later mapped to command lookup table for it's execution. Here Sphinx library contains humongous number of words and processing them would reduce in time efficiency of the system. Thus to avoid this, right now we have only taken a pool of phonetics that are needed to recognize the commands for OkEclipse+.

2. **PDE:** We have used Eclipse's PDE (Plugin Development Environment) to develop the OkEclipse plugin. The Plug-in Development Environment (PDE) provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins, fragments, features, update sites and RCP products. PDE also provides comprehensive OSGi tooling, which makes it an ideal environment for component programming.
We have also made use of "PDE Build" to facilitate the automation of plug-in build processes. Essentially,

PDE Build produces Ant scripts based on development-time information provided by, for example, the plugin.xml and build.properties files. The generated Ant scripts, can fetch the relevant projects from a CVS repository, build jars, Javadoc, source zips, put everything together in a format ready to ship and send it out to a remote location (e.g., a local network or a downloads server).

3. **Java:**

12 ARCHITECTURE

Here instead of implementing the whole system again we are leveraging the architecture used by the first team and appending our architecture to the existing system. To implement automatic error fixing using speech recognition we needed to get the existing command used by Eclipse to do the same. When the user take cursor to line with compile time error and presses keyboard shortcut "Cytl+1", Eclipse suggests some solutions to the error and user can select and implement any one of them. This whole process is accomplished by Eclipse through methods of QuickFixProcessor and QuickAssistProcessor.

As you can observe from figure[], we have detected the voice command using CMU Sphinx and checked it with the command lookup table formed by the first team. Here, in the lookup table we have added "FIX" keyword for error correction. Once "FIX" is detected we are doing the same process that the user has to do manually by pressing keyboard shortcut "Cytl+1". OkEclipse+ will deal with QuickFixProcessor and QuickAssistProcessor for compile time error detection and correction and execute it using "core.commands.Command".

Figure[] shows the architecture and working of CMU Sphinx library that is used for speech-to-text for command recognition.

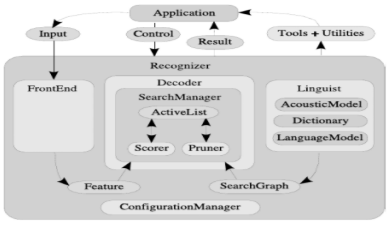


Figure 9: Architecture-2

13 PLAN OF WORK

The Gantt chart in Figure 10 shows the anticipated schedule of our project. The whole project is divided into major tasks and each task is given a duration. This chart will be used by the teammates to look at regularly. This will help us divide the major tasks into smaller sub-tasks and keep the track of timeline.

- In the first week we plan to do the detailed analysis of the already implemented project, which would include tasks such as understanding the technology stack and reviewing the implemented code.
- Next step would be brainstorming to think of new ideas which would augment the currently implemented application.
- We will also evaluate the user-needs and feasibility of our ideas. For this we planned to roll out surveys to around 30 developers.
- We would then increase the pool of commands in the existing application. We have got a good response for the same from the stakeholders.
- Lastly, we want to implement Stanford NLP library in the plugin, that would help with context recognition.

14 EVALUATION PLAN

To evaluate the working of our project and also to know on first hand that how users would react to OkEclipse+ we decided to have a small group of Computer Science students use our plugin.
Here, to keep the evaluation fair and accurate we decided to judge it based on matrix consisting of number of errors against the time taken by users to solve them in two ways i.e. manually by using keyboard shortcut "ctrl+1" and using OkEclipse+ voice command. The results of this evaluation can be observed from the table[].
From the above observations we derived two formulae which can give approximate time taken for each according to number of errors.

$$Manually = Numberoferrors * time/error$$

$$Fixcommand = C + Numberoferror * time/error$$

The it can be seen that there is huge difference between number of errors and time taken to solve them. The reason for this trend is that when the user is manually solving errors using keyboard shortcut, he/she will have to locate the error, scroll down till there and then move the cursor and press keyboard shortcut. While for OkEclipse+, the user will have to just say the command "FIX" once and error handling will be done automatically by the plugin. When there is only one error the user has to just press the keyboard shortcut while for the voice command the voice will be first detected and then execution of the command will be done which will take some minimum time. So, we can conclude that voice command will definitely prove to be much more efficient as the number of errors increases. Thus, fulfilling our motive to increase time efficiency of users while using IDE for coding.

No. of errors	Manual(secs)	'FIX' cmd(secs)
1	2	2.5
5	12	4.3
10	29	4.2
15	39	5.4

The following graph[] represents the values shown in table[].

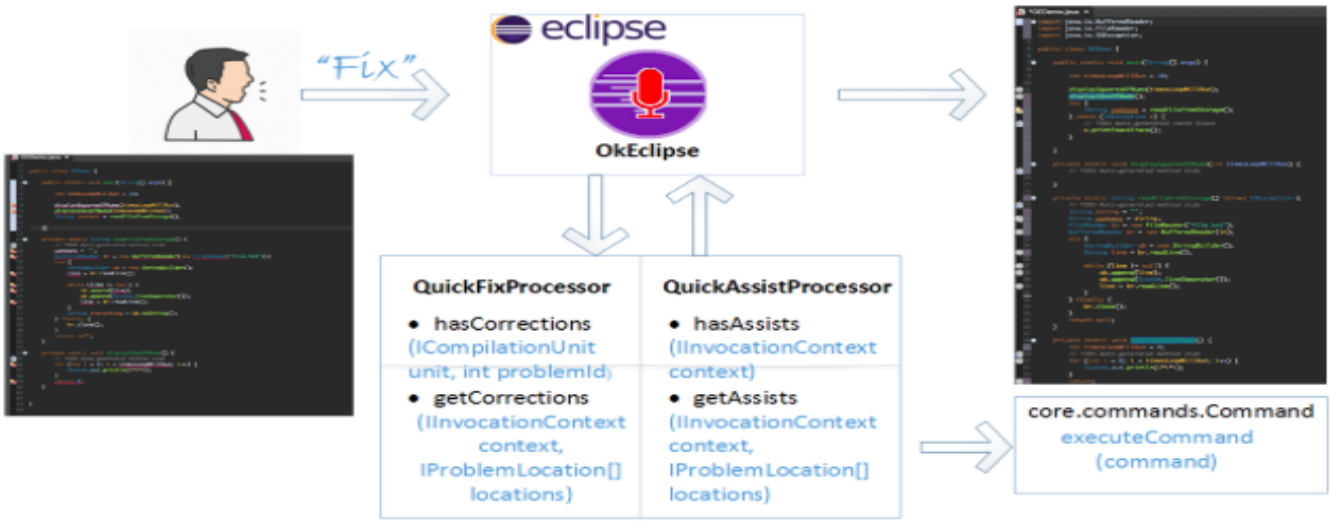


Figure 8: Architecture

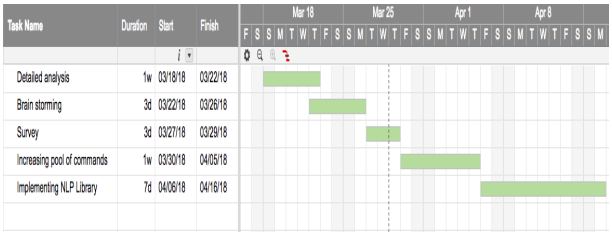


Figure 10: Plan of Work

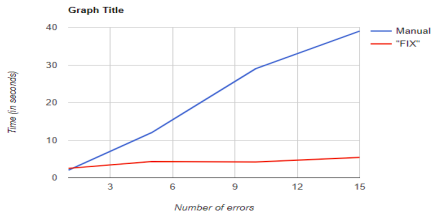


Figure 12: Report

15 EVALUATION RESULT

To check the acceptance of of our application among users we created a google form to take survey, which was taken by 21 of our fellow classmates.The survey questions were aimed at knowing how users felt about the improvisation of okEclipse. Following section elaborates our survey questions with reason why that question was a part of the survey and also the statistics of the response that we got. The survey included sections for email ID and name of the user but as mentioned by professor we didn't make them required as to ensure the right to anonymity of the users.

1. Which one take lesser time to solve an error: FIX command or without FIX command.
The question was intended to compare the response times of resolving an error manually and by resolving the errors using FIX command.

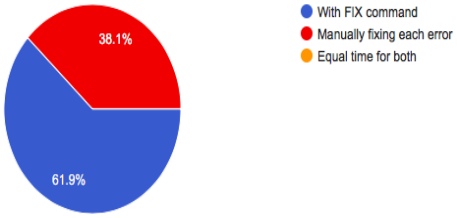


Figure 13: Graph 1

2. Was the system able to recognize and detect your command first time or you had to make multiple attempts? If multiple attempts then please provide us with a number.
Here although we didn't change anything in the base that the first group provided and the work of voice detection was done very thoroughly by them, we wanted to make sure that the command we added was detected properly or not. This question was intended for the same.

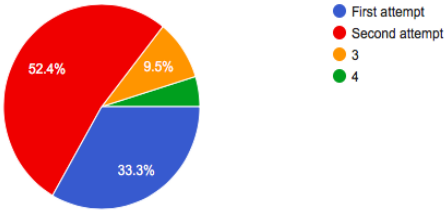


Figure 14: Graph 2

We were glad to see that almost 85 percent user commands were detected in either first or second attempts.

3. Rate the ease of usage of Eclipse after adding this plugin

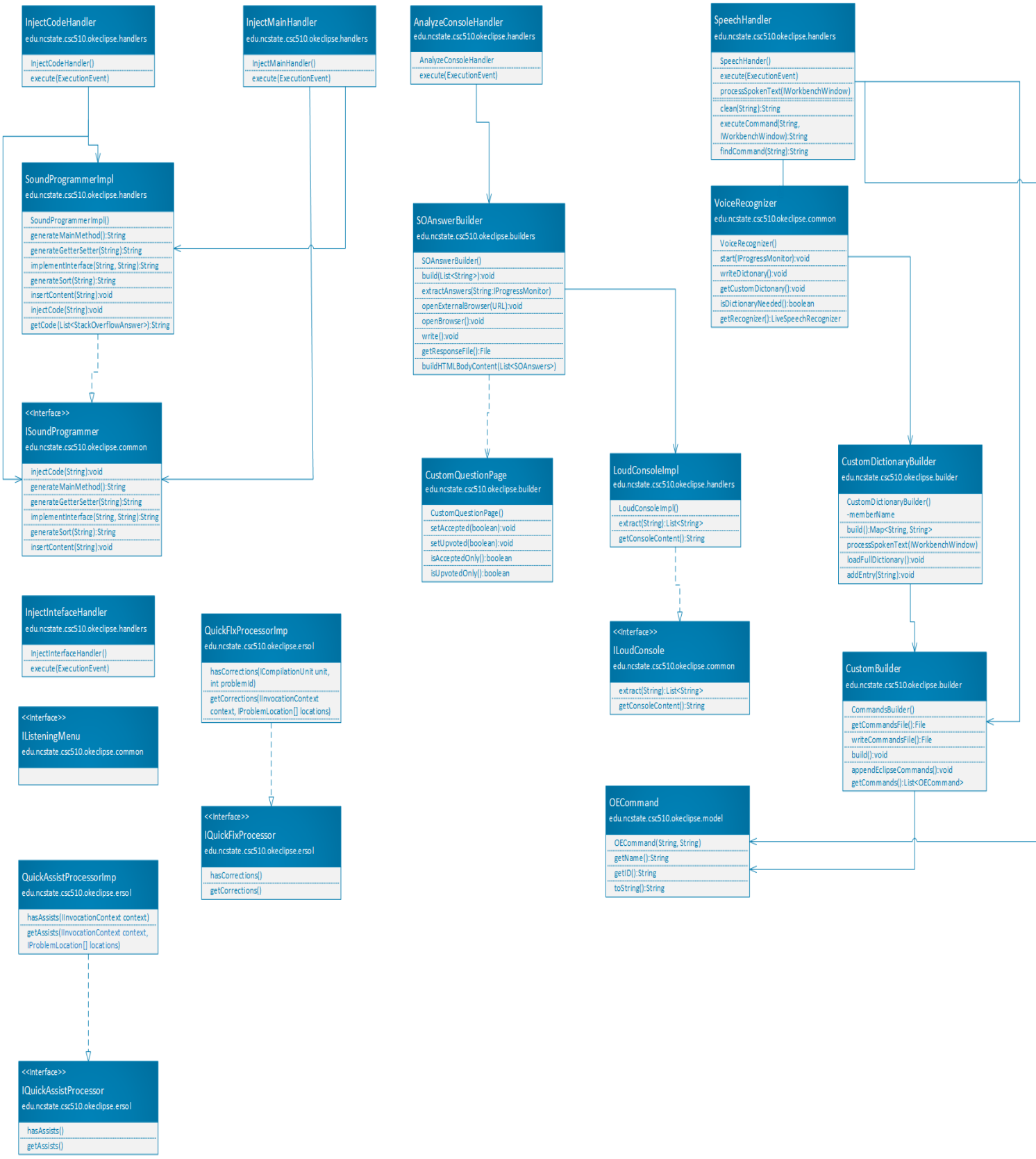


Figure 11: Class Diagram

Our motive behind the whole idea was to make resolution of compile time errors easy for users of IDE. To know whether users were finding it easy to use our plugin or rather they would prefer the keyboard shortcut, we decided to ask the above question. Here we asked them to rate the ease of usage as they don't have to answer a 'yes' or 'no' question but can express their minds.

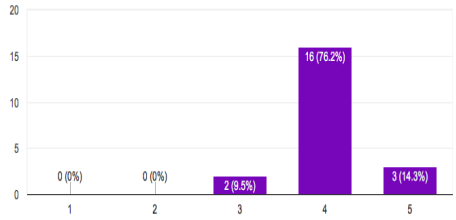


Figure 15: Graph 3

As we can see we got a rating of 4 for ease of usage by 76.2 percent surveyors.

4. Rate your experience with OkEclipse+

This question was just to know reaction of users regarding the idea of automatic error resolution. This was again a question carefully designed with a rating for answer so as to avoid putting user in a spot with a 'yes' or 'no' option.

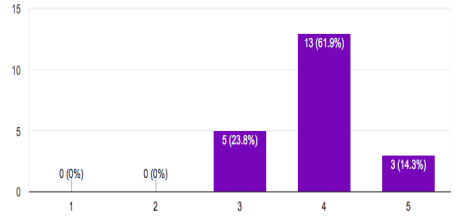


Figure 16: Graph 4

More than 75 percent users gave us a rating of 4 and 5 which concludes that the users were satisfied with our work.

16 FUTURE SCOPE

A speech plugin for Eclipse can have large number of future scopes. We have explored few of them which are described below. Also we can consider the future scopes mentioned by the first team like incorporating speech plugin for other IDEs and creating a verbose life cycle i.e. accepting not only keywords but rather getting statements from users and extracting the keywords.
Following future scopes can be explored to enhance OkEclipse+:

1. Here extensions can be added to allow implementation of multi-lingual support for speech plugin and also speech synthesis.

2. We can also consider the approach that we write our own speech recognition grammar which is customized for user-defined words or phrases related to Eclipse.

3. Provide easy navigation through all the functions provided by the system, such that any activity handled by the system can be done without the need of mouse or keyboard. For example "search", accessing "start" button and many other support functions that are necessary for IDE.

4. Parallelizing speech as well as keyboard inputs. Schneiderman's research [10], he observes that cognitive resources decrease significantly when a user is restricted to speech only and multi-modal usage often results in increase in available cognitive resources. He reports that humans find it difficult to speak and think at the same time, yet using a keyboard in tandem with speech system resulted in 15-30 percentage increase in speed. Thus, parallelizing spoken output with keyboard typing can be considered.

17 CONCLUSION

Currently Ok Eclipse is able to identify commands using voice recognition. However it does not help in solving runtime/compile time errors. Hence, we have proposed a method for error handling in Ok Eclipse using voice recognition as one of the solution to improve the user experience. In this report we have defined the problem and our motivation. Manageable requirements have also been stated along with a suitable plan of work that we presume will be the most effective way to evaluate the intended goal of our project.

References

[1] Shaik, S., Corvin, R., Sudarsan, R., Javed, F., Ijaz, Q., Roychoudhury, S., Bryant, B. (2003, October). SpeechClipse: an Eclipse speech plug-in. In Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange(pp. 84-88). ACM.

[2] Begel, A.,& Graham, S. L. (2006, September). An assessment of a speech-based programming environment. In Visual Languages and Human-Centric Computing, 2006. VL HCC 2006. IEEE Symposium on (pp. 116-120). IEEE.

[3] Diaz, Jaime, and Raiza Muniz. "Voice Recognition System." (2007).

[4] Bell, Donald. "UML basics: An introduction to the Unified Modeling Language." The Rational Edge (2003).

[5] Meyer, Stephenie. Eclipse. Little, Brown Books for Young Readers, 2007.

[6] Writing an Evaluation Plan. https://www.brown.edu/research/conducting-research-brown/preparing-proposal/proposal-development-services/writing-evaluation- plan.

[7] B.Boehm. Software Productivity and Reuse. IEEE Computer Society. Sept. 1999.

[8] University of Calgary Kent Beck et al. JUnit - Unit testing framework. http://junit.org. [Online; accessed 1/30/2018]. 2018.

[9] Eclipse Foundation. Eclipse Next-Generation IDE. https: //www.eclipse.org/che/. [Online; accessed 1/30/2018]. 2018.

[10] S.L. Graham A.Begel. An assessment of a Speech-BAsed Programming Environment. Visual Languages and HumanCentric Computing, 2006, VL/HCC, IEEE Symposium.

CHIT NUMBERS

- JTV
- XYG
- HGI
- LXM
- ANG
- DQH
- LTQ
- PTZ
- ABW
- KAJ
- QYO
- TQD
- GHR
- QWD
- XIM
- HXK
- JHQ
- ROL
- FYK